UNITED STATES PATENT APPLICATION

For

A METHOD AND SYSTEM FOR INTEGRATING NETWORK-BASED FUNCTIONALITY INTO PC APPLICATIONS AND DOCUMENTS

Express Mail label No.: EJ606935351US
Date of Deposit: September 29, 2000

MORGAN & FINNEGAN, LLP
345 PARK AVENUE
NEW YORK, NEW YORK 10154-0053

A METHOD AND SYSTEM FOR INTEGRATING NETWORK-BASED FUNCTIONALITY INTO PC APPLICATIONS AND DOCUMENTS

BACKGROUND OF THE INVENTION

The present application claims priority under 35 USC § 119 to Provisional Patent Application, serial number 60/217,361, filed on July 11, 2000, for "System and Method for Integrating Network-based functionality into PC Applications", and is hereby incorporated by reference.

Field of the Invention

The present invention relates generally to a method and system for providing network-based functionality into personal computer applications and documents, and more particularly, to a system and method for embedding network-enabling objects into PC documents and providing network-based functionality thereto.

Description of Related Art

There exists today a rich class of end user applications generally termed as "productivity applications", used to create, view and modify personal computer (PC) documents. Readily available PC productivity applications include any PC application for creating a variety of documents on a PC, for example, creating spreadsheets for managing and manipulating numerical data, creating drawings and illustrations, and for creating mechanical blue prints using computer-aided design (CAD) programs. Documents created and managed by such applications are typically stored on the same PC (local disk) or a file server on the local network. The content

of these documents is generally self-contained; therefore all content presented to the end user is stored in the document itself.

On the other hand, there exists a vast quantity of information and variety of services that are hosted remotely, accessible via a network. A web browser, for example, is a convenient vehicle for accessing such content and even for hosting simple applications that are downloaded from the network which work with such remote information and services. Such simple applications may manifest themselves as more advanced web pages, Java applets, browser "plug-ins", or the like. The extended functionality supported by such simple applications often attempts to recreate a segment of functionality found in an existing desktop application. However, none of these additions to web browsers (or other network clients) incorporate the full functionality of commonly available productivity applications. Moreover, as additional functionality is deployed to web browsers in attempts to raise their level of functionality, bandwidth requirements increase causing the application to become more prohibitive to download.

In general, documents managed by productivity applications are self-contained and do not allow viewing or manipulation of remote content or access to functionality provided by remote systems and servers on a network. Certain PC applications allow users to embed web page links or e-mail address links into a document created by the underlying PC application. These PC applications allow users to launch Internet clients by clicking on the link. However, it is important to note that the result of performing such an operation from within the PC application is for the underlying PC application to separately launch the corresponding Internet client, essentially passing the end user to a distinctly new environment. Since the content being viewed in the Internet client, whether a web page or a new email, cannot be manipulated by the

original underlying PC application, the user is forced to either work within either the PC application or the Internet client; thus, users are limited to one environment or another.

Common mechanisms for deploying and utilizing network technologies include web browsers, media players, e-mail clients, and/or other Internet clients. Similarly, the content presented and manipulated by these various network technologies cannot be integrated fully into the PC applications' environment such that the content can then be manipulated by the native functionality and tools resident within the PC applications. As a result, while such browsers, media players, e-mail clients, and other network technologies allow access to information on the Internet or other networks, they do not provide the functions and operations that are typically performed by PC applications. For example, while a browser allows a user to search the World Wide Web, one cannot directly perform ad-hoc analysis of stock data retrieved from the Internet using models running within a spreadsheet application. As a result, end users must manually move the content into a PC application geared towards such analysis.

Attempts have also been made to receive and collect remote content from external sources from within a PC application for use toward particular specialized tasks. For example, certain personal financial management applications retrieve stock data from the Internet and bring the data into local views to augment existing content. Unfortunately, once the information has been "imported", the local view remains static, instead of updating itself automatically, as the remote content changes. Moreover, while the ability to integrate this imported information with local content or distributed content from other sources exists in certain limited situations, no means are provided by the PC application to extend the access capabilities to include other sources of remote content. Users are therefore limited solely to the types of information/data provided by the remote information source.

At the same time, other PC applications collect and extract data/information from multiple sources and bring the data/information into some central location, such as spreadsheets, databases and the like. These PC applications provide the capability to extract data/information from one or more sources and marshal the data/information into one central location; however their functionality is primitive. For example, a user may be limited to extracting particular data (such as a table of data, but not an individual row or a single paragraph) from a public or static web page, or unable to supply required credentials to gain access to a private web site. Additionally, the functionality exists solely within the application at hand (a user cannot for example apply a query tool for one application to seamlessly retrieve data for another application). These existing tools are merely provided as limited enhancements to the existing application, but fails to provide a platform for development of a wide range of enhanced functionality for said application.

Furthermore, constructing distributed applications that manipulate remote content but execute within a network client such as a browser represents an onerous task at best. The remote content cannot be saved locally and manipulated while not connected to the remote source and later reused when connected to the original remote location. Additionally, as mentioned above, as the level of functionality approaches that of the full desktop productivity application, the resulting browser application becomes prohibitively large for typical distribution via Internet download. Lastly, once such a web application has been constructed, it fails to operate with a mix of both local and remote content.

Moreover, existing desktop applications may provide a method of extending their native capabilities via a scripting interface; however these scripting environments are often inferior to typical robust development platforms. When attempting to construct solutions in

these environments, developers must resort to more complex code and debugging methods, thereby additionally raising development costs and reducing the availability of developers who may have the necessary skill sets to implement such solutions. Therefore a need exists for a more robust development platform, allowing a developer or end user to quickly build and deploy applications that expose network functionality integrated with PC productivity application functionality.

As such, on one hand, existing PC applications fail to incorporate Internet and/or other general network-based access capabilities as an integral part thereof; on the other hand, Internet clients fail to incorporate the full scope of PC application functionality therein. As a result, these PC applications cannot be utilized for network-based operations and functionalities. Furthermore, available PC applications fail to provide solid mechanisms for incorporating distributed and remote information into their respective documents. For example, while the ability to receive network-based information, such as stock quote streams, in real time, directly into a spreadsheet or to receive live video feeds within a document of today's PC applications, exists today in certain limited situations, coupling dynamic remote content with documents that are automatically updated when the remote content changes requires functionality currently unavailable in such productivity applications. Thus, a user is denied access beyond the traditional functions provided by the standard PC application and/or Internet clients, and cannot create documents that leverage these network-based features.

In view of these disadvantages, there is a need for a system that allows seamless and integrated access to both local as well as distributed content and information. There is a further need to extend the network access and utilization capabilities of existing PC applications. In addition, there is a need to provide a general as well as extensible framework for extending the

network access and utilization capabilities of existing PC applications. Furthermore, there is a need for a system to create, manage and utilize documents that contain a mix of immediate as well as distributed content. There is further a need for managing PC documents containing PC applications so that PC documents can behave as canned applications themselves. There is yet a further need to enhance the packaging and delivery of documents containing PC applications so that they are suitable for web application deployment. Additionally, there is a need for managing access to and use of local content based on remote settings and controls.

SUMMARY OF THE INVENTION

The present invention overcomes the above-mentioned disadvantages. One aspect of the invention provides a general system for augmenting existing PC applications, such as word processors, spreadsheets and the like, with network-based functionality therein. Users are provided with the ability to utilize and leverage the resources of the network/Internet from within PC documents opened in their PC applications. For example, the augmented PC application of the instant invention allows the user to browse the Internet, read and write emails, send and receive instant messages, perform page scraping, receive stock quotes in real time, access remote databases over the Internet, or perform other similar network-based functions from within PC documents, and be able to use their PC application functionality against both local and remote content in concert.

In one embodiment of the present invention, a personal computer document with network-based functionality is disclosed. The personal computer document is used with the personal computer application augmented with a network-functionality software that is resident in computer memory. The personal computer document is coupled with network-enabling

objects that are configured to provide network-based functionality to the personal computer document.

The invention described herein supports integration of a wide range of network-based user interfaces, content, data, and functionality into personal computer applications and personal computer documents. The networks that can be used in conjunction with this technology include the Internet as well as other networks.

The invention described herein enables the development, deployment, and use of these network-based services and documents with network-based functionality. The invention allows integration of a wide range of network-based, including Internet-based, user interfaces, content, data, and functionality into the augmented PC applications and also into PC documents used or produced by such augmented PC applications. Thus, the invention enables professional developers as well as end users to develop PC application-based, network-enabled documents and applications by using augmented versions of PC applications with which they are already familiar. This simplifies and improves the development of network-based applications or documents.

The present invention also provides a client/server architecture wherein one or more users may connect to a network, comprising one or more independent servers, for receiving network-based functionality within their personal computer applications and personal computer documents.

The present invention may include an architecture that involves four software components. The first component is a document-independent, run-time client-side code, through which a personal computer application is extended by network-enabling software that facilitates access to network-enhanced services and functionalities from within the application

environment. The second component of the architecture is the enhanced personal computer document, which includes network-enabling objects. The third component of the architecture is a design tool for creating the enhanced personal computer document. The fourth component of the architecture is server-side code that manages interactions between clients and Internet resources and interactions among multiple end users.

These aspects and other objects, features, and advantages of the present invention are described in the following Detailed Description which is to be read in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1A is a block diagram of a personal computer document in a personal computer application augmented by a PCA extender;

FIG. 1B is a block diagram of another embodiment of a personal computer document in a personal computer application augmented by a PCA extender;

FIG. 1C is a schematic diagram of a network-enabling object in accordance with the present invention;

FIG. 2 is a schematic diagram of a personal computer application extended by a client extender in accordance with one embodiment of the present invention;

FIG. 3A is a schematic diagram of one embodiment of network-enabling software for use by software developers;

FIG. 3B is a schematic diagram of one embodiment of network-enabling software for use by lay users;

FIG. 4 is a schematic flow diagram depicting the process for providing network-enabling software to a personal computer application;

FIG. 5 is a block diagram of a personal computer application that provides users with built-in tools for creating documents with network-based functionality;

FIG. 6 is a schematic diagram of a personal computer document in accordance with the present invention;

FIGS. 7A-7D are schematic diagrams of different states of communication between the network-enabling objects in the PC document with one or more controls installed on the PC;

FIG. 8 is a schematic flow diagram depicting the process for creating a personal computer document with network-enabling objects embedded therein;

Fig. 9 is a schematic flow diagram depicting the process for launching network-based functionality with a personal computer document;

FIG. 10 is a schematic block diagram depicting the client/server system of one embodiment of the present invention for providing network-based functionality with a personal computer document;

FIG. 11A-11B are screen-shots depicting two embodiments of a personal computer document in accordance with the present invention;

FIGS. 12A-12E are screen-shots of a personal computer document augmented with developer tools in accordance with the present invention;

FIG. 13 is a schematic block diagram depicting the client/server system of an embodiment of the present invention for developing and accessing a web page as an embedded object that is produced using a spreadsheet.

FIG. 14 is a schematic flow diagram depicting the process for developing and accessing a web page as an embedded object that is produced using a spreadsheet application;

FIG. 15 is a schematic flow diagram depicting the process of the operation by a user to use a web page created in accordance with the process of Fig. 14;

FIG. 16 is a schematic block diagram depicting the client/server system of an embodiment of the present invention that facilitates the cooperative writing and editing of newspaper articles and other written works;

FIG. 17 is a schematic flow diagram depicting the process of using the system of FIG. 16 by a software developer;

FIG. 18 is a schematic flow diagram depicting the process of using the system of FIG. 16 by a writer;

FIG. 19 is a schematic flow diagram depicting the process of using the system of FIG. 16 by an editor;

FIG. 20 is a schematic block diagram depicting the client/server system of an embodiment of the present invention that allows an illustrator or another author to create, distribute, and license illustrations and other works of authorship;

FIG. 21 is a schematic flow diagram depicting the process of using the system of FIG. 20 by an illustrator; and

FIG. 22 is a schematic flow diagram depicting the process of using the system of FIG. 20 by a customer.

With reference to the following detailed description, the aforementioned drawings will be described in greater detail below. The leading reference numeral in each drawing indicates the first drawing in which the reference numeral is introduced (e.g., elements 320 and 1140 are introduced in figures 3 and 11 respectively).

DETAILED DESCRIPTION

The present invention relates to a system and method for providing network-based functionality to a PC document. Initially, a PC application (PCA), which is a software program used by a PC for a particular desired task, is augmented with network-enabling software that allows users to provide new features to the PCA. The network-enabling software of the present invention enables users to generate, modify and utilize PC documents with embedded objects. The objects can work in conjunction with the network-enabling software to launch and utilize network-based functionality from within PC documents. The present invention enables the use of a wide array of public and private network-based content and services including, but not limited to, content and web services and other Internet-based content and services, within a PCA, such that these services can be viewed and used within such applications, and within the documents associated with these applications.

It should be noted that the present invention, even when described in connection with the Internet specifically, may be applied equally to local area networks (LANs), private "Intranets," hybrid public-private "Extranets," virtual private networks (VPN), and other networks.

The services that can be provided and used within a PCA include web browsing, electronic mail (e-mail), instant messaging, digital rights management (DRM), remote database access over the Internet, and other Internet-based services, as well as combinations of these content and services. These services may be provided from within a word processor such as Microsoft WORD, a spreadsheet such as Microsoft EXCEL, a design program such as AUTOCAD, a desktop publishing program such as QUARK EXPRESS, or any other PCA.

According to the invention, once the network-enabling software has been installed on a personal computer, the network-enabling software may automatically load when the augmented PCA is loaded. Users of the augmented PCA may load existing PC documents or create new ones with the augmented PCA. Once a PC document is open within the augmented PCA, the user is able to make any of the standard operations thereto, such as creating, adding or modifying content therein. Furthermore, the augmented PCA allows the user to embed network-enabling objects in the PC document by a plurality of mechanisms.

According to the invention, the network-enabling objects embedded in a PC document allow users to launch network-based operations from within the PC document when the associated PCA has the network-functionality software running alongside. Such network-enabling objects may allow users to perform web browsing, electronic mail (e-mail), instant messaging, remote database access over the Internet, DRM, and/or a plurality of other network-based operations consistent with the purpose of the present invention from within PC documents. The invention also enables integration of PCAs as well as PC documents with remote applications, databases, and data sources, where communication between them may occur over a plurality of network types.

According to the invention, both fixed and customizable objects may be placed within individual PC documents to provide network-based functionality thereto. The network-enabling objects can either be visible to the user within the document or hidden. The network-enabling objects may contain data, UI elements, and code such as Visual Basic scripts. According to the invention, the network-enabling objects can present the user with specific options, and act in response to user-provided information and choices, and/or they may take network-related actions automatically, driven solely by software commands.

As noted above, the actions taken by the network-enabling objects placed within PC documents can include any combination or sequence of a wide array of Internet-related actions, including but not limited to checking or sending e-mail, checking or sending Internet-based voicemail, querying remote databases over the Internet, requesting or updating a web page, invoking a remote application over the Internet, establishing a real-time data feed with a specific Internet data source, DRM, or sending a real-time message.

It should be noted that the ability to make use of these services and functionalities is not restricted to existing PCAs and PC documents. Such services and functionalities can also be provided through future PCAs not yet developed, future PC documents not yet developed, and through augmented or enhanced future versions of existing applications and document types, or through wireless devices.

According to another embodiment, the PC document may be provided with an installation facility to install the network-enabling software on a PC where the PC document resides but where the underlying PCA has not been provided with the network-enabling software of the present invention. The invention discloses a system for automatic installation of the network-enabling software so that new PC documents with network-enabling objects may be created, manipulated, enhanced and supported.

The invention further discloses the packaging of network-based functionalities via software development kits (SDKs) and application programming interfaces (APIs) usable by professional PCA developers. It will be appreciated that the functionalities provided by the invention, the architecture in which they are embedded, and the APIs through which they can be utilized, greatly simplify and improve the development of new, network-enabled PCAs and PC

documents. These functionalities can also be provided for network-specific documents such as e-mail messages that are processed by SMTP/POP e-mail clients.

In general, the present invention may include an architecture that involves four software components. The first component is a document-independent, run-time client-side code, through which a personal computer application is extended by network-enabling software that facilitates access to network-enhanced services and functionalities from within the application environment. The second component of the architecture is the enhanced personal computer document, which includes network-enabling objects. The third component of the architecture is a design tool for creating the enhanced personal computer document. The fourth component of the architecture is server-side code that manages interactions between clients and Internet resources and interactions among multiple end-users.

The runtime client-side code may be configured as one or more distinct sub-components. For example, one sub-component may constitute generic client-side code that handles aspects of system operation common to all applications, while a second sub-component may be application specific (e.g., the portion of the code that controls the application's unique user interface). This runtime support code at the client can be developed using any of a number of programming languages, such as C, C++, Java and Visual Basic. One of the functions of the runtime client-side code is to wrap and package low-level transport protocols such as TCP/IP and HTTP into elements that can be manipulated and that provide specific, high-level functional capabilities, such as wrapping SMTP to provide easy-to-use email capabilities, wrapping HTTP Web page access and parsing to allow easy access to live information on web pages. The wrapping is done in a layered fashion so that end-users can create and/or manipulate the

elements without having any programming capabilities whatsoever, yet developers can leverage scripting or lower-level programming to access more advanced facilities.

The second functionality of this runtime client-side code is to integrate with the hosting PCA. This integration falls generally into three categories. The first category simply provides a seamless runtime environment so that these services run and are available from within the application process. This makes the packaged services available for calling from within scripts in enhanced documents and from application code itself, and is generally enabled by running at least part of the support code as an add-in to the application itself. The second category of integration is at the user interface (UI) level, where the enhanced functionality elements are exposed as operations on toolbars and menus, and as objects in enhanced documents themselves. This integration is enabled through a combination of standard windowing facilities on the client platform, and use of the application and document object models to allow integrated rendering of the UI elements and handling of associated events. This, for example, enables an end-user to drag an email button that has been provided on a toolbar into a document, such that the button is rendered as part of the document and that when the button is pressed a pre-defined message is emailed to the author of the document or other specified recipient. The end-user would need no programming ability to perform this embedding and configuration, having only to fill in the content of the message and the name or email address of the recipient. The third integration category relates to embedded applications, and involves controlling the behavior of the hosting application itself when an embedded application has been placed in "run mode". Document-based applications are generally not designed to behave as application containers for embedded applications, and when used in this manner must be managed carefully to prevent unwarranted operations by end-users. This management or control

can involve blocking certain operations, prevention of the embedded application from being rearranged or reconfigured, or modifying and/or augmenting various ordinary tasks such as saving a local copy of the embedded application enhanced document. This control is achieved again by packaging some or all of the runtime support code as an extension (such as an add-in or a plug-in) so that it runs in close cooperation with the hosting PCA, and in working with the PCA extension to change the PCA behavior. The PCA extension can change the PCA behavior by a combination of handling the entry points and events, overriding handling by the PCA itself, and extending the PCA object model through new, custom code.

In this runtime environment, an augmented PCA may allow an end-user to download and open PC documents within the PCA's native environment. In one embodiment, the document may constitute a web page, or a PCA version thereof, downloaded from a web server. The augmented desktop application and enhanced document together form an Internet or network-enabled application that facilitates access to remote services and functionalities while retaining functionality inherently provided by the desktop application. Access to distributed services may be via any network or protocol.

The enhanced PC document includes Internet-enabling objects which can be document specific and placed into a document such as a PCA-based web page and the like by either a developer or end-user. The UI of such objects can include buttons, lists, menus, and forms. Their functionality can include combinations of Internet or network-based functions such as sending and receiving data, sending and receiving e-mail messages, causing documents to be posted to web sites, and other functions enabled by the technology. The functionality of such objects is specified by custom-developed software, which can be associated with the object's UI, and then activated by software events and/or user-issued commands. Once written, these objects

can be separately transmitted via the Internet and stored on client machines for use by developers and end-users.

In certain embodiments of the present invention, enhanced PC documents created by a developer can be defined as embedded applications, consisting of three defined elements, namely a wireframe, population data and form data.

The wireframe includes static content and scripting code that is generally not modified during runtime within a PCA residing on an end-user's system. The static content also includes the layout configuration of the document viewed by the end-user, such as fonts and/or colors. The script code allows end-users or developers to extend the built-in functionality of the PCA or simply create macros containing often-used packaged sequences of operations (for example, repeatedly creating the same customized chart from a set of data). In the context of the wireframe, the script code may contain code to assist in supporting the present invention.

The population data is dynamically generated data used to populate specified areas within a wireframe during a desktop application's runtime. Typically, such data is supplied by a server and is designated "read only", but where appropriate the data may be editable, or can be pushed back to the server for other custom processing. Population data may be used in a variety of ways, including to populate fields, strings, and pick lists for display to the user, or transparently as calculation data operated on by the application. It may also be defined to be a function of information maintained in the associated wireframe, user input, or both. It may typically be generated or retrieved upon actions such as startup, connection, or form submission, or a manual refresh request or in real-time in response to remote data changes.

The form data is data which is specified or manipulated at the client by end-users during runtime. Form data is usually one-way data transmitted from client to server. But in

some circumstances it may be desirable to read such data from a server (for example, as cached prior information to pre-populate a user form or as smart defaults for a user form) to pre-fill forms within a wireframe. Form data includes data items, such as data entered into text box prompts (for example, asking for a user's username and password for authentication) and parameterized data entered into defined queries (such as stock ticker symbol as a parameter in a we b query).

The design tool for creating enhanced PCDs augments a PCA so that a software developer is provided with functionality that is similar to a special-purpose PCA that is oriented towards the design of complex enhanced PC documents, services and applications. In the design-time environment, an augmented PCA can provide a developer with tools for creating the distributed documents described above.

It should be noted that support for the design-time environment is similar to the packaging and functionality described above for the general runtime support except that the integration with and control of the containing application is generally more extensive. In addition, the elements and tools that are exposed to a developer are generally more detailed and in many cases will involve scripting or code to achieve the greatest level of functionality and control. The principal difference in functionality in the developer's design time tools and environment from an end-user's authoring environment is that the former provides extensive controls targeted at defining and managing the behavior of the containing application, while the latter is focused on creating and managing elements in enhanced documents and/or directly accessing the distributed functionality and services of the runtime itself. The design-time support code will generally be packaged as one or more separate add-ins and/or tools from the standard runtime code. Furthermore, the runtime code can be divided along functional

boundaries into multiple, separately-distributed pieces as well. For example, this runtime could be sub-divided into components or libraries that are completely independent of any particular enabled application (perhaps as a collection of COM objects), an application-specific add-in that provides base end-user tools and support for enhanced documents, and a second application-specific add-in that provides for application and environment control for embedded applications, where the enhanced document is actually intended to be an embedded application and have a distinct design mode and run mode. The development environment may then be packaged as one or more additional add-ins distinct from the above.

The design tool allows the developer to design the distributed PC documents described above. The developer may be provided with tools that allow him or her to embed Internet or network-related functionality (e.g., automated e-mail notifications) in the PC document, control the UI that will be presented to the end-user when the document is opened, define population-data sources and how the data is displayed, define business rules, and various other functions within the scope of the present invention.

The developer may also create parallel browser-based views of these distributed documents. The choice of whether to download a browser-based view or desktop-application-based view of a page may be controlled by preferences set by the end-user and stored at a server. The developer may also wish to augment the browser-based version of the page so that it provides some of the additional functionality included in the PCA based version. Examples of such network-functionality may include scripting for form-data posting, HTML-based lightweight grid handling, and inserting meta-tags into the browser-based view of a page for managing later wireframe synchronization with the desktop application version.

Since some developers may wish to augment the browser-based version of the web application still further, the invention provides the ability to retain such modifications and enhancements through subsequent desktop application-to-augmented-HTML exports, and push modifications back to the desktop application version where appropriate. A typical example of HTML-side augmentation is inserting client-side scripting to add functionality to the UI beyond that provided by a browser (for example, ornate hover buttons). Additional desktop application-side augmentation might involve a business analyst defining macros and calculations linking various data items within the application.

The invention provides for server-side components that manage interactions between clients and Internet resources and/or interactions among multiple end-users, the latter greatly facilitating client peer-to-peer operations. Specific services can be characterized by the kind of server support they require. For example, E-mail notifications require basic SMTP mail services. Semi-standardized services can be leveraged for functionality such as remote database access, messaging, conferencing and the like. Custom remote services can be leveraged for accessing proprietary information feeds (such as real-time stock quotes), document archives and repositories, corporate workflow systems and the like. For these latter categories, the present invention adds value by providing a generalized framework for finding and accessing the remote services, and for opening standardized connections for communication and data transfer to the client PCA.

With reference to the figures, various embodiments of the present invention will now be described in greater detail. FIG. 1A provides an illustration of a system in accordance with the present invention. A PCA 100 is provided with a PC document 110. In accordance with the present invention, the PCA 100 is not limited to any particular task, form or vendor.

According to the present invention, the PC is not limited to any particular type, and is operable on any machine with a suitable operating system having GUI facilities. The operating system for use in the present invention is also not limited to any particular type, and may include Microsoft® Windows®, Macintosh, UNIX, Linux, Net BSD, FreeBSD, and the like. These functionalities can also be provided to applications running on Personal Digital Assistant (PDA) devices, cellular phones and other "light clients" connected to the Internet via wireless communications.

According to the invention, the PCA 100 is augmented so that it is capable of developing, using, accessing, manipulating, and displaying network-enhanced documents. Such an augmented PCA 100, which allows for the creation and use of network-enhanced documents, essentially be termed as an application within an application. The present invention enables access to Internet-based services, data, and functionalities within network-enhanced documents, which are developed, edited, and viewed via the augmented PCA 100. The invention provides for tools, pre-defined scripts, and supporting runtime code that make construction of such enhanced documents in the desktop application's native format simple enough for end-users, yet flexible and comprehensive enough for developers to rapidly construct complex, powerful applications that leverage both Internet/network services and the client-side functionality of the particular desktop application. Where possible, the technology leverages a built-in script execution capability provided by the PCA 100, such as Visual Basic for Applications facilities provided by Microsoft. Thus, this system provides a powerful environment for developing and using enhanced documents with rich and familiar client-side functionality.

The PC document 110 is an object that is typically created, used, manipulated, augmented and saved by the underlying PCA 100. For example, a spreadsheet is the document

where the underlying PCA 100 may be a spreadsheet application, such as Microsoft® Excel or Corel® Quattro®. The PC document 110 may be provided with embedded or associated code that provides access to distributed, usually Internet-based, services, data and functionalities residing on multiple computer systems connected via the Internet or other network. These services and functionalities may be made available via the Internet using the typical network protocols, such as hypertext transfer protocol (HTTP), FTP, SMTP (e-mail), streaming media, or RMI.

PCA 100 is provided with a PCA extender 120. The PCA extender 120 enables users to augment the PCA 100 with network-based functionality. The PC document 110 is provided with network-enabling objects 130 that provide network-based functionality to the PC document 110. The network-enabling objects 130 may be used to provide any network-based functionality from within the PC document 110. For example, the network-enabling objects 130 may be used for browsing the Internet, screen scraping, sending and receiving electronic mail (e-mail), instant messaging, chat, internet telephony, streaming media, stock quote retrieval, web page update detection, workflow, conferencing, licensing, automatic notification of a certain event, usage monitoring, replication and synchronization and other similar network-based tasks and content.

The network-enabling object 130, which is placed inside the PC document 110, can perform a wide range of network-based functions, as specified by a developer. Such network-enabling objects 130 can include UI elements, such as buttons, pick lists, and menus, program code, such as Visual Basic scripts that perform functions, and data elements. These objects can be placed into the PC document 110 by developers and end-users, where they remain as part of the PC document 110 structure, upon saving the PC document 110.

Coupling the PCA extender 120 to the embedded network-enabling object 130 provides one or multiple network-based capabilities, such as web browsing and e-mail within the PCA 100. As a result, the PCA 100 serves as the UI to the Internet and other networks, which eliminates the need to rely on separate Internet-specific client applications, such as web browsers (e.g., Netscape Navigator), e-mail clients (e.g., Microsoft Outlook or Eudora), and media players (e.g., distributed by Real Networks). Furthermore, the provision of web access within the PCA 100, employing the native data types and facilities of the PCA 100 for the representation of web pages, allows users to use the web site from within the PCA 100. This allows users to have access to functions such as browsing pages, navigating through hyperlinks or image maps, entering data in forms, and a variety of other network-based functions.

According to one embodiment of the invention, the PCA extender 120 may interact with one or more Component Object Model (COM) components on a local PC to support and provide additional functionality within the PCA as well as PC document, as shown in FIG. 1B. COM is a software architecture that allows components made by different software vendors to be combined into a variety of applications. COM defines a standard for component interoperability and provides the underlying support for Object Linking and Embedding (OLE) items and ActiveX® controls to communicate with other OLE objects or ActiveX® controls. OLE is a framework for a compound document technology, available from Microsoft®. OLE is a set of APIs to create and display a compound document. An ActiveX® control is a software module based on the COM architecture, which enables a program to add functionality by calling ready-made components that blend in and appear as normal parts of the PCA. ActiveX® controls are typically used to add user interface functions, such as 3-D toolbars, a notepad, calculator or even a spreadsheet. In many cases, COM provides the underlying services of

interface negotiation, life cycle management (determining when an object can be removed from a system), software licensing, and event services (putting one object into service as the result of an event that has happened to another object).

According to the present invention, the network-enabling objects 130 communicate with the PCA extender 120 to enable the document to receive network-based functionality therein. The PCA extender 120 communicates with the COM component(s) 140 to allow the embedded network-enabling objects 130 in the PC document 110 to provide additional functionality to the PC document.

FIG. 1C provides an illustration of one embodiment of the network-enabling object 130 in accordance with the present invention. The network-enabling object 130 is used to launch the network-based functionality within the PC document 110. The network-enabling object 130 is embedded in the PC document 110 so that the functionality of the network-enabling object 130 becomes at integral part of the PC document 110. The network-enabling object 130 comprises two main components, initialization code 190 for initializing and launching the network-based functionality prior to runtime, and runtime code 192 for providing the network-based functionality during runtime. As a result, the initialization code 190 causes the launching of the network-based functionality, while the runtime code 192 enables a user to continuously receive network-based functionality within the PC document 110.

It should be noted that the initialization code 190 plays no role during runtime, and remains static/dormant during runtime of the network-based functionality. According to one embodiment, the runtime code 192 can be provided with all the necessary functions or routines for enabling the PCA extender 120 to read the necessary information and then initialize and launch the network-based functionality. According to another embodiment, the PC document

110 can be embedded with code that includes some of the necessary functions/routines to initialize and launch the network-based functionality, and the initialization code 190 can be provided with the bare minimum routines to merely initialize the network-enabling object 130.

According to one embodiment, the network-enabling object 130 may be provided with a UI 194 that makes the network-enabling object 130 visible to the user. It should be noted that providing the UI 194 for the network-enabling object 130 is strictly optional, since a number of network-enabling objects 130 may be provided such that they are hidden from the user's view while still providing network-based functionality.

FIG. 2 provides an illustration of a PCA 100 in association with the network-enabling software 210 of the present invention and other generic add-in(s) 220 that may be supplied by one or more third-party vendors or developers. As noted above, the PCA 100 is not limited to any particular task, form or vendor. Accordingly, the PCA 100 may be a spreadsheet application, a word processing application, a desktop publishing application, a presentation graphics tool, multimedia development tools (for music, video and the like), a computer aided design tool, a project management application, a client database manager, a personal finance organizer, a personal information management tool, a browser, an e-mail client, an instant messaging client, a chat client and other software. Furthermore, the PCA 100 may be provided by any vendor, including Microsoft®, Corel®, Sun Microsystems and the like.

The PCA 100 comprises native PCA code 230. The native PCA code 230 includes code that makes the PCA 100 operational and provides the functionality and features that are standard for the PCA 100. The native PCA code 230 generally comprises the code that creates and provides the user interface (UI) 232 for the underlying PCA 100.

The native PCA code 230 exposes a native object model 235. The native object model 235 provides a description of an object architecture, including the details of the object structure, interfaces between objects and other object-oriented features and functions. The native object model 235 enables the use of add-ins 220 and network-enabling software 210 so that the functionality of the add-ins 220 and network-enabling software 210 can seamlessly be integrated with the original functionality and features of the PCA 100.

The native PCA code 230 may also include an Application Program Interface (API) 240, which may be a part of the native object model 235. The API 240 provides a language and message format used by the PCA 100 to communicate with the operating system or some other system or control program or communications protocol. The API 240 is implemented by custom authored components, which leverage the API 240 and manipulate the PCA 100. The API 240 also provides a method for extending the PCA 100 by the PCA extender 130 or other COM objects, allowing software developers to author custom functionality; as a result, the API 240 allows manipulation of the operation of the PCA 100 from an external source, as opposed to from within the PCA 100 and/or the PC document 110. The API 240 provides developers a blueprint of how to manipulate the PCA 100 by publishing a consistent set of interfaces.

The native PCA code 230 provides a native script engine 250. The native script engine 250 is a facility in the PCA 100 that can execute interpreted languages, such as Visual Basic, JScript scripts and/or other types of scripts. The native script engine 250 allows the script code in the PC document 110 to execute and manipulate the PCA 100 environment.

The native PCA code 230 may include a PCA extension enabler 260 which allows and facilitates the use of add-ins and/or plug-ins with the PCA 100. The PCA extension enabler

260 enables add-ins and/or plug-ins that have been added to the PCA 100 to be loaded and run automatically when the PCA 100 is launched. Generally, the PCA extension enabler 260 is a manifestation of the API 240. As such, the PCA 100 may either have an API 240 or a PCA extension enabler 260.

Using the available features of the PCA 100, one can create a new document 110 or open an existing document 110 to perform operations thereon. The network-enabling software 210 allows users to embed network-enabling objects into the document 110, which will be described in greater detail below. Once the network-enabling objects are embedded in the document 110, the network-enabling software 210 allows a user to use the network-enabling objects and receive the network-based functionality in their documents 110, as well as use the standard functions and features provided by the PCA 100.

FIG. 3A provides an illustration of one embodiment of the network-enabling software 210. The network-enabling software 210 includes the PCA extender 120, which provides any standard PCA 100 with a one or more network-based capabilities.

The PCA extender 120, which provides runtime support, includes network-functionality services 310 as well as application services 315. The network-functionality services 310 may include session management, security and location management that provide network-based functionality, as well as some combination of the individual network-functionality services. The network-functionality services 310 provide runtime support for the network-based functionality. The network-functionality services 310 expose an API with methods than can be called by the network-enabling objects 130 and scripts from within the PC document 110 to use one or a plurality of network-based functionalities. The network-functionality services 310 also includes the base functionality for packing away script code of the

PC document 110 for safe and easy network transport as well as unpacking a script code 720 (FIG. 6) of the PC document 110 when the PC document 110 is opened.

The application services 315 may include XML parsing and other handling, event handling and service management, query routing, document packaging, caching, data initialization and persistence, command routing and other similar services, as well as some combination of the individual application services. The application services 315 provide runtime support for the embedded applications and/or network-enabling objects 130 in the PC document 110.

According to another embodiment, the PCA extender 120 is a COM component. The PCA extender 120 is deployed as a dynamic link library (DLL) file, which is not launched directly by the user but may be called by the PCA 100 for receiving network-based functionality. As such, the PCA extender 120 may also be used by any application running system-wide, in addition to being accessible within the PCA 100. It should be noted that designing the PCA extender 120 as a DLL file is just one method of deployment, and various other manners of deployment exist within the scope of the present invention. As such, each functionality of the PCA extender 120 may be provided as a separate routine, class or function in the DLL file, which may be packaged to include one or more of the network-based functionalities listed above.

It should be noted that the PCA extender 120 may be provided with just the network-functionality services 310 or just the application services 315 or a combination of the two. Furthermore, the scope of the present invention is not limited by the types of services included in the network-functionality services 310 and/or the application services 315.

The network-enabling software 210 may further be provided with developer tools 320, which provide a plurality of functionalities to a developer who wishes to create an enhanced

PC document 110. The developer tools 320 may automatically be invoked upon the launching of the PCA 100. According to one embodiment, the developer tools 320 have a UI for display to the developer in the form of a toolbar. The developer tools 320 also allow manipulation of an information model 790 (FIG. 7D) for the PCA 100, including handling of wireframe data, population data, and the queries used to retrieve the population data. The developer tools 320 provide software developers with the ability to construct network-enabled PCAs and create end-user tools and utilities, and network-enabling objects 130. The developed functionality may be used for various network-based capabilities. It should be noted that the functionality is developed in accordance with the information model 790 (FIG. 7D).

The developer tools 320 are configured to provide various sets of functionality, enabling a developer to build and deploy a compatible network-based PC document. The functionality may include layout utilities, forms management, query management, data modeler, a packaging assistant, and other miscellaneous utilities and operations that assist in network-based activities, along with any combination of one or more of the enumerated functions.

According to one embodiment, the developer tools 320 are created and packaged as a COM component. The developer tools 320 package is deployed as a DLL file. As such, the developer tools 320 may also be used by any application running system-wide, in addition to being accessible within the underlying PCA 100. According to another embodiment, each functionality of the developer tools 320 may be provided as a distinct COM component that can be installed by the software developers at their own wish. According to yet another embodiment, groups of functionalities of the developer tools 320 may combined into single COM components that can be installed by software developers at their own wish. It should be noted that designing

the developer tools 320 as a DLL file is just one method of deployment, and various other manners of deployment exist within the scope of the present invention.

The PCA extender 120 and the developer tools 320 are integrated or coupled to the PCA 100 through the PCA extension enabler 260. This integration may be achieved through standard methods of extensibility. It should be noted that the PCA 100 may make itself extensible through a variety of means. For example, Microsoft's products can be extended through what they term as an "add-in" model, where the COM object extenders, such as the developer tools 320 and the PCA extender 120, are defined as add-ins at design time, through, for example, the use of a wizard or by implementing a required programming interface.

FIG. 3B provides an illustration of a second embodiment of the network-enabling software 210 for use by end-users. The network-enabling software 210 is provided with the PCA extender 120 that provides the PCA 100 with a plurality of network-based capabilities, as well as a plurality of user tools 330.

As noted above, the PCA extender 120, which provides runtime support, includes network-functionality services 310 as well as application services 315. It should be noted that the PCA extender 120 may be provided with just the network-functionality services 310 or just the application services 315 or a combination of the two. Furthermore, the scope of the present invention is not limited by the types of services included in the network-functionality services 310 and/or the application services 315.

The network-enabling software 210 may further be provided with a plurality of user tools 330. The user tools 330 provide the ability for a user to receive network-based functionality within the desired PCA 100. The user tools 330 enable end-users to embed network-enabling objects 130 and functionality into their PC documents 110 through the use of

toolbars, user wizards and the like. In other words, the user tools 330 may comprise a UI that blends in with the UI 232 of the PCA 100, wherein the users can utilize the UI of the user tools 330 to seamlessly embed the desired objects 130 for the network-based functionality in the PC document 110.

According to one embodiment of the invention, the user tools 330 are configured to provide various optional features, such as page scraping, stock quotes in real time, and other miscellaneous utilities and operations that assist in network-based activities within the spirit of the present invention. As a result, the user tools 330 enable users to embed live stock feeds, create objects to send and receive e-mails, selected web pages to embed in the PC document 110, and other similar network-based functions. The user tools 330 allow embedding of network-enabling objects 130 that use the network-functionality services 310 at runtime to achieve the network-based functionality.

As noted above, the user tools 330 often work in conjunction with the PCA extender 120 to provide said network-based functionality.

It should be noted that the user tools 330 are intended for the end user. On the other hand, the developer tools 320 are intended for the developer. The PCA extender 120 may be utilized by both the user tools 330 as well as the developer tools 320. As such, if for example one created a set of functionality geared at lawyers, called "lawyer tools," the lawyer tools may leverage some of the packaged functions of the PCA extender 120 in addition to the functionality contained therein. It should be noted that all components of the network-functionality software 210 work in conjunction with the network-enabling objects 130 in the PC document 110.

According to one embodiment, the user tools 330 are created and packaged as a COM component. The user tools 330 are deployed as one DLL file. According to another

embodiment, the user tools 330 may be provided as distinct COM components that can be installed by software developers at their own wish. It should be noted that designing the user tools 330 as a DLL file is just one method of deployment, and various other manners of deployment exist within the scope of the present invention.

The present invention also provides users with the ability to incorporate extended services 340 designed by the third-party software developers. According to one embodiment, the third party developers may create network-enabling objects using the developer tools 320 (FIG. 3A). According to another embodiment, the extended services 340 may be created by any generic tool for developing COM components. The use of extended services 340 provides the flexibility of later augmenting a PCA 100 with new network-based functionality that may be created or authored in the future, while still being designed in accordance with the information model 790 (FIG 7). This allows developers the opportunity to leverage the existing functionality available in the PCA extender 120 or even the user tools 330 or developers tools 320.

The PCA extender 120, the user tools 330 and the extended services 340 are integrated or coupled to the PCA 100 through the PCA extension enabler 260. This integration may be achieved through standard methods of extensibility. It should be noted that the PCA 100 may be extended through a variety of means. For example, Microsoft's products can be extended through an "add-in" model, where the COM object extenders, such as the user tools 330, the extended services 340 and the PCA extender 120, are defined as add-ins at design time, through, for example, the use of a wizard or by implementing a required programming interface.

FIG. 4 depicts one embodiment of the process for providing custom network-based functionality to a PCA 100 from the network-enabling software 210. The COM component is installed on a PC that has the PCA 100 running thereon, and may be used in

conjunction with the documents opened in the PCA 100. The COM component enables the network-enabling objects 130 to receive the network-based functionality in the opened document 110.

At step 410, the custom functionality is authored, which may leverage functionality found in PCA extender 120, the developer tools 320 and the user tools 330. There are various mechanisms to create the custom functionality, such as using Visual Basic® development language to create ActiveX objects, programming a component in the C++ language, and the like. At step 420, the developer of the custom functionality enters certain custom registration information into the object created. The custom information may include, for example, properties such as an object's fixed name, display name, and Global Unique Identifier (GUID). This custom registration information is needed to conform the created custom functionality to work correctly with the rest of the invention's architecture. The custom registration information is used to provide a common way for all of the pieces to be able to communicate with each other. With correct registration information, a custom set of functionality can link itself to, and make use of, the PCA extender 120. The PCA extender 120 then becomes aware of the custom functionality and the services provided, and can manage them appropriately. It is important to conform the created custom functionality appropriately so that the PCA extender 120 can be aware of and manage the custom functionality. In one embodiment, the custom functionality will make use of the services offered by the PCA extender 120 as well as use the PCA extender 120 to communicate with network-enabling objects 130 within the PC document 110.

As shown in step 430, once the authored custom functionality has the appropriate registration information, the functionality is packaged as a DLL file with the desired

configuration. Thus, if the developer wishes to deploy a plurality of DLL files in an installation program, the DLL files would be packaged together with an installation manager. It should be noted that the developer can also set other properties on the network-enabling object 130, such as threading models, debugging levels, and the like.

At step 440, the DLL for the appropriately packaged custom functionality is provided to the user. The user may obtain the custom functionality in a variety of different ways, such as in an installation disk or Compact Disk (CD), or via download from a web site. The obtained DLL for the custom functionality is installed on a PC, as shown in step 450. The installation takes place in accordance with the developer's designed method, such as running an install program to automatically install the DLL on the PC in the appropriate location.

At step 460, the custom DLL registers with the operating system registry, which allows the DLL to operate/function correctly under the COM architecture. For example, if the DLL is being installed on Microsoft® Windows® 3x, the DLL registers with the Windows® registry. The registry provides a secure, unified database that stores configuration data in a hierarchical form, which allows system administrators to easily provide local or remote support, using administrative tools. The registry also allows developers to store their own configuration data for their custom applications.

At step 470, additional information is read from the DLL by a service manager that resides in the PCA extender 120. This is the information originally entered in step 420 by the developer, dictating the rules for the manner in which the custom functionality is to be utilized.

Finally, in step 480, the service manager of the PCA extender 120 makes the registered DLL available to the PCA 100. It should be noted that the DLL has the necessary

properties, run-time information and other information needed. At this point, the service manager in the PCA extender 120 knows that a new set of functionality exists and how to offer the functionality to other system components. Once the PCA extender 120 is installed, the service manager is available.

Once the necessary PCA extender 120 and developer tools 320 have been installed, the software developer may use these tools to author custom network-based functionality that can be made available as extended services 340.

It should also be noted that end-users will utilize the aforementioned process for installing the necessary user tools 330 as well as extended services 340. Once the necessary PCA extender 120, user tools 330 as well as extended services 340 have been installed, the users may access network-based functionality in their documents 110 opened within the PCA 100.

According to the invention, and as discussed above, two logical pieces of code are required for operation of the invention. The first piece is the embedded network-enabling objects 130 and a network-enabling code 630 (FIG. 6). The second piece is the network-enabling software 210, including the PCA extender 120 and/or user tools 330 and/or developer tools 320. While in one embodiment, the network-enabling software 210 is an external piece of software that augments the PCA 100, as discussed above, in another embodiment, the network-enabling software 210 can be embedded into the PCA as an integral part thereof.

FIG. 5 provides an illustration of a PCA 500 which is configured to provide users with network-based functionality as an integrated feature thereof, in accordance with the present invention. The PCA 500 may be augmented by the use of one or more generic add-in(s) and/or plug-in(s) 220 that may be supplied by third-party vendors.

The PCA 500 comprises a native PCA code 520. The native PCA code 520 includes code that makes the PCA 500 operational and provides the functionality and features that are standard for the PCA 500. The native PCA code 520 generally comprises the code that creates and provides the UI 522 for the underlying PCA 500.

The native PCA code 520 may also include a PCA extension enabler 530 which allows and facilitates the use of add-in(s) and/or plug-in(s) with the PCA 500. The PCA extension enabler 530 is a facility that allows generic add-in(s) to plug into the PCA 500 for adding new functions, operations or features that were not originally available in the PCA 500.

The native PCA code 520 may expose a native object model 540. The native object model 540 includes a description of an object architecture, including the details of the object structure, interfaces between objects and other object-oriented features and functions. The native object model 540 enables the use of an add-in 220 or any other macro or extender running on the PC so that the functionality of the add-in 220 and/or other macros or extenders can seamlessly be integrated into the PCA 500. The native PCA code 520 may also include an API 544 to allow the PCA 500 to interact with other programs for expanding the standard features or providing additional functionality thereto. According to one embodiment, the API 544 is an integral part of the native object model 540.

The native PCA code 520 may provide a native script engine 550. The native script engine 550 is a facility in the PCA 500 that executes macros or any other extenders that are to be run along with the PCA 500.

The native PCA code 520 is provided with code that provides network functionality. The code for network functionality 560 comprises a plurality of network-based utilities. The code for network functionality 560 may be configured to handle application

services, such as XML parsing and handling, session management, security, event handling and service management, location management, query routing, document packaging, caching, data initialization and persistence, command routing and other similar functions and operations that assist in network-based activities. In addition, the network-based utilities are configured to provide various optional utilities that are used to create PC documents 110 with network-based functionality therein. The network-based utilities may include layout utilities, forms management, query management, data modeler, packaging assistant, page scraping, stock quotes in real time, and other miscellaneous utilities and operations that assist in network-based activities within the spirit of the present invention.

Using the available features of the PCA 500, one can create a new document 110 or open an existing document 110 to perform operations thereon. The code for network functionality 560 allows users to embed network-enabling objects into the document 110, which will be described in greater detail below. Once the network-enabling objects are in the document 110, the code for network functionality 560 allows a user to use the network-enabling objects and receive the network-based functionality in their documents 110, in addition to the non-network functionality and features provided by the PCA 500.

FIG. 6 provides an illustration of one embodiment of a PC document 110 in accordance with the present invention. The PC document 110 comprises document code 610 and content 615. The document code 610 includes all computer readable instructions necessary to provide the PC document 110 with its look and feel, as well as provide support for any functionality embedded in the PC document 110.

The document code 610 includes a script code 620. The script code allows end-users or developers to extend the built-in functionality of the PCA or simply create macros

containing often-used packaged sequences of operations (for example, repeatedly creating the same customized chart from a set of data). The script code 620 also may provide the ability to interact with ActiveX controls and other COM components on the PC to utilize their provided functionality within the PC document 110.

The document code 610 may further include network-enabling code 630 for providing network-based functionality within the PC document 110 at launch-time by initializing and launching the network-enabling object 130 to provide network-based functionality. According to one embodiment, the network-enabling code 630 is unpacked and read by the PCA extender 120 at launch-time, and after that the network-enabling code 130 is not used, accessed, and/or read during run-time. In other words, during run-time of the network-based functionality, the network-enabling code 630 plays no role, and is akin to not being present. The network-enabling code 630 is unpacked (i.e., read) at load/launch time by the PCA extender 120 to support the functionality of the PCA extender 120.

The network-enabling code 630 contains computer readable instructions for allowing the initialization and/or launch of the network-based functionality as well as any other functions within the scope of the present invention. According to one embodiment, the network-enabling code 630 may include a plurality of routines to provide different operations and functionality. According to another embodiment, the network-enabling code 630 may be a plurality of distinct modules, separately packaged in the document code 610. The network-enabling code 630 may be programmed to include support for query instructions, layout instructions, data modeling, and various other features that may augment the PC document 110 with additional network-based functionality.

According to one embodiment, the network-enabling code 630 is packed together in a secure location within the document code 610 of the PC document 110. In the event that the network-enabling code 630 is packed, it must be unpacked prior to use of network-enabling code 630 for providing network-based functionality. Furthermore, the network-enabling code 630 may be encrypted with a suitable encryption algorithm to ensure that the network-enabling code 630 is tamper-proof, as well as to restrict utilization of the underlying network-based functionality to the appropriately registered modules. According to one embodiment, the network-enabling code 630 is packed for storage and/or transmission over a network or the Internet.

It should be noted that when the PC document 110 with network-based functionality is stored and/or uploaded, the PC document 110 can have a mix of packed and unpacked network-enabling code 630. The packed portion of the network-enabling code 630 requires unpacking prior to run-time of the network-based functionality.

The content 615 in the PC document 110 includes data 660 and network-enabling object(s) 130. The data 660 may include static data 670 as well as dynamic data 680. Static data 670, once entered, maintains its state and is often only altered by a user. Dynamic data 680, on the other hand, may be added and/or altered at any time, possibly without any active intervention by a user. For example, the PC document 110 could be configured for allowing the dynamic data 680 to be altered in real-time by content from the Internet or other networks by providing a real-time stock feed into the PC document 110.

It should be noted that according to one embodiment, the content 615 may also be provided as a mix of packed and unpacked content in addition to the network-enabling code 630. For example, where it is desired that the user not be able to access certain pieces of content, the

content may in fact not get completely unpacked at runtime. Similarly, it is possible to not unpack part of the network-enabling code 630 to deprive an unlicensed user of its functionality, and the still-packed code 630 would be inert and not executable at runtime. Additionally, it is possible to pack the content 615 in an encrypted format.

The network-enabling object 130 may be embedded as part of the content 615 of the PC document 110. As part of the invention's distributed architecture, the network-enabling object 130 may be able to communicate with the script code 620. Through the script code 620, the network-enabling object 130 may communicate with the PCA extender 120, and the PCA extender 120 then communicates with the network-enabling code 630. It should be noted, that the network-enabling object 130 is also capable of interacting directly with the PCA extender 120.

According to one embodiment, the network-enabling object 130 may be a self-contained packaged set of functionality. According to another embodiment, the network-enabling object 130 will operate in conjunction with the PCA extender 120 and other custom functionality that persists on the client, such as user tools 330, developer tools 320 and/or other add-ins.

According to the invention, the developer may leverage the distributed architecture and decide how "heavy" to make the network-enabling object 130 portion of the code, by providing all the requisite code for receiving network-based functionality as part of the network-enabling object 130. The developer can decide the amount of functionality to be put into the network-enabling object 130, versus putting the network-based functionality as a custom add-in. The best balance will depend on the particular functionality being deployed, and perhaps

even on deployment issues such as code size for add-in code being downloaded from the Internet for installing on the client.

FIG. 7A-7D provides an illustration of different states of communication between the PC document 110 with one or more controls external thereto. According to an embodiment of the present invention, the PC document 110 comprises a wireframe 705, which has content 615, native script code 620 and network-enabling code 630. The content 615 comprises user interface (UI) 710, which has information regarding formatting and layout for the data 660 and the objects 130 therein. The UI 710 essentially provides the look and the feel of the content 615. For example, the UI 710 controls the fonts, formatting and the layout of text in the PC document 110. The content 615 further includes configuration 720, which enables and facilitates the user's viewing thereof. For example, the configuration 720 may allow a user to decide the zoom-in level of the PC document in a Microsoft® Excel spreadsheet.

As noted above, the content 615 includes static and/or dynamic data, depending on the choice of the user. In addition, the content 615 is provided with network-enabling objects 130 for launching network-based functionality from within the PC documents 110.

According to the invention, the native script code 620 provides the ability to interact with ActiveX® controls and other COM components on the PC to utilize the network-based functionality within the PC document 110. The native script code 620 permits the PCA 100 to leverage its core functionality and permits the PCA 100 to provide network-based functionality from within the PCA 100 environment. While there are various network-based functionalities that can be provided, they may include communications services that allow a user to communicate and interact with others, streaming services that allow a user to receive and play

streaming content, data services that allow a user to retrieve, manipulate, and view data retrieved over a network, as well as various other services within the scope of the present invention.

According to one embodiment of the present invention, the native script code 620 allows any of the network-enabling objects 130 in the PC document 110 to interact with the PCA extender 120 running alongside the PCA 100. More particularly, external components communicate with the network-enabling objects 130 via the PCA extender 120. While this is one mechanism of operation, it should be noted that the present invention also allows the PCA extender 120 to directly communicate with the network-enabling objects 130, without requiring any intervention from the native script code 620, according to another embodiment of the present invention.

The network-enabling code 630 facilitates the providing of network-based functionality within a PC document 110. The network-enabling code 630 contains computer readable instructions for launching network-based functionality as well as any other functions within the scope of the present invention. The network-enabling code 630 may be programmed to include support for query instructions, layout instructions, data modeling, and various other features that may augment the PC document 110 with additional network-based functionality. As noted above, the network-enabling code 630 may be packed in a secure location within the document 110, in which case it must be unpacked prior to its use. Furthermore, the network-enabling code 630 may be encrypted.

According to the invention, the native script code 620 is capable of communicating with the PCA extender 120 or other external COM components 780 or ActiveX controls 770. According to the invention, there are different models according to which the network-enabling objects 130 may interact with the PCA extender 120 to provide network-based

functionality, as shown in FIGS. 7A-7D. Accordingly, a developer may extend and customize the PCA extender 120 in a variety of ways to provide any additional type of customized functionality.

When designing additional ActiveX 770 and/or COM controls 780, the developer can have them work in conjunction with network-enabling objects 130, allowing the PCA 100 to become a distributed application. The developer has the flexibility to design and provide network-based functionality in different ways, since they can utilize the PCA extender 120, or they can provide the network-based functionality via the native script code 620. According to one embodiment, a part of the distributed PCA manifests itself as the network-enabling object 130 and another part takes the form of an ActiveX control 770. According to another embodiment, another part of the distributed PCA could be a COM control 780 that leverages functionality available in the PCA extender 120.

FIG. 7A provides an illustration of the present invention at launch-time of the network-based functionality. Initially, network-based functionality is launched when the PCA extender 120 reaches into the PC document 110 and unpacks (i.e., reads) the network-enabling code 630. The information from the network-enabling code 630 is stored in the PCA extender 120 which allows it to provide the appropriate network-based functionality at runtime. It should be noted that the PCA extender 120 is not supported by the other external COM components 780 or ActiveX controls 770, initially. Furthermore, in this embodiment, the PCA extender 120 does not need to communicate with the native script code 620 to unpack the network-enabling code 630.

As shown in FIG. 7B, according to one embodiment, the network-enabling object 130 communicates with the PCA extender 120 to provide network-based functionality in the PC

document 110. According to this embodiment, the network-enabling object 130 is provided with the necessary code to interact with the PCA extender 120 to launch, obtain and utilize the network-based functionality within the PC document 110. It is possible for the PCA extender 120 to interact with the COM component(s) 780 to seek additional functionality not available in the PCA extender 120, as indicated by the instructions contained in the network objects code. It should be noted that the network-enabling code 630 play no role in the process. According to this embodiment, the native script code 620 is not necessary for providing the network-based functionality.

As shown in FIG. 7C, according to another embodiment, the PCA extender 120 communicates with the network-enabling object 130 to provide network-based functionality in the PC document 110. Furthermore, the PCA extender 120 may interact with the COM component(s) 780 to seek additional functionality not available in the PCA extender 120. Based on the operations and features of the PCA extender 120 and/or the COM components 780, the native script code 620 may be provided with the requisite information to allow it to communicate with the network-enabling objects 130. According to one embodiment, the native script code 620 can communicate directly with the ActiveX 770 and/or COM controls 780 to provide additional functionality not available through the PCA extender 120. As a result, the network-enabling objects 130 may provide network-based functionality in accordance with the PCA extender 120 as well as the ActiveX 770 and/or COM controls 780.

As shown in FIG. 7D, according to another embodiment, the network-enabling object 130 communicates with the PCA extender 120 to provide network-based functionality in the PC document 110. Furthermore, the information from the unpacked network-enabling code 630 that is read by the PCA extender 120 provides the requisite information to the native script

code 620, and creates and provides the necessary information to an information model 790. Where the network-enabling code 630 is provided packed, the PCA extender 120 unpacks the network-enabling code 630 prior to launch to read the necessary information therefrom and create the information model 790. The information model 790 is a holding area that houses information about the functions the PC document 110 is supposed to support at runtime, and the methodology for running the functions. For example, the information model 790 might contain a query definition of how to retrieve stock quote information, namely the server to look for, the data parameters to retrieve, and the mechanism for handling the resulting data sets.

The information model 790 interacts with the network-enabling objects 130 to provide the appropriate network-based functionality within the PC document 110.

Since network-based functionality may be distributed between network object 130 and Com component 780, the desired network-based functionality can be authored by a developer according to the best suited structure. A developer may choose to place more functionality in COM component 780, resulting in a "lightweight" network-enabling object 130. This allows easier distribution of the PC document 110 once the base COM component 780 has been installed.

FIG. 8 is a schematic flow diagram depicting the process for creating a personal computer document with network-enabling objects embedded therein. In general, the objects 130 may be embedded into PC documents 110 in several ways. They can be placed into documents by developers, and the documents can then be posted on a web site or e-mailed to users. When coupled with the ability to control and manage the behavior of the containing application, this basically provides for the development of powerful, distributed "embedded applications" that are essentially defined by the combination of the containing application and

the enhanced document within it. Alternatively, the network-enabling objects 130 can be made available to end-users and stored on end-user systems. End-users can then place objects personally into individual PC documents 110 they create or receive. This capability can be used in combination with other Internet or network functions.

The present invention may also be used to provide professional developers with a tool set for creating PCA-based services and applications, such as "PC Application Web Sites." While certain development capabilities may best be provided through a specific, separate application program for developers using the technology, many capabilities may be provided to developers through an extended version of the target PCA itself. Thus, for example, this will allow a high fraction of web site development to be conducted using the familiar, functionally rich, and domain-specific environment of a PCA rather than traditional web site development tools or lower level development environments such as C or Java. This will facilitate the workflow processes associated with web site development, by permitting partially completed PCA-based web sites to be e-mailed, viewed, and edited by multiple developers and end-users, using augmented versions of a single PCA. Moreover, where business rules can be defined using the built-in script and/or native functionality of the document and desktop application, parts of the PCA-based web site can essentially be coded directly by domain experts (i.e., end-users) rather than passing specifications off to a developer for implementation, thereby potentially reducing development time and cost.

Referring to FIG. 8, at step 810, the PCA 100 is launched, along with the network-enabling software 210. According to one embodiment, because of the appropriate configuration of the network-enabling software 210, it automatically launches when the PCA 100 is launched. At step 820, the PC document 110 is opened. According to the invention, the

PC document 110 may be new and completely blank, initialized through a pre-defined template, or a pre-existing document.

Once the PC document 110 is open, data may be created therein, as shown in step 830. For example, content for a word processing document may be entered by the user. As noted above, the data may be static or dynamic. In step 840, network-enabling objects 130 are embedded as part of the content 615 of the PC document 110. There are a variety of ways in which users may be allowed to embed the network-enabling objects 130 in the PC document 110. According to one embodiment, the user is provided a toolbar as the UI of the network-enabling software 210, from which the user can simply choose the desired network-enabling object 130, and then drag and drop in the appropriate location of the PC document 110.

As one embodiment, the system allows a user to encrypt the network-enabling code 630, in step 850. In step 860, the network-enabling code 630, which may or may not be encrypted, is placed in the document code 610 area of the PC document 110. According to one embodiment, the network-enabling code 630 may be packed in a hidden area so that it is invisible to the user. According to another embodiment, the network-enabling code 630 may be compressed before being placed in the document code 610 area of the PC document 110.

Finally, in step 870, the PC document 870 is saved on the local system. As the above discussion shows, the saved PC document 110 comprises data as well as network-enabling objects 130 and network-enabling code 630. During runtime, the network-enabling objects 130 and the native PCA code 230 interact with the network-functionality software 210 to provide network-based functionality in the PC document 110.

FIG. 9 provides one embodiment of the process for loading the PC document 110 with network-based functionality therein. At step 910, the PCA 100 is launched by a user. As

depicted in box 920, once the PCA 100 has been launched, the user opens a standard document for the launched PCA 100. For example, if the launched PCA is Microsoft® Excel, then the open document will be an Excel spreadsheet or any other compatible Excel file type.

At step 930, the installed PCA extender 120 detects the opening of the PC document 110. At step 940, the installed PCA extender 120 inspects to determine if the PC document 110 has the appropriate network-enabling objects 130 and the network-enabling code 630 therein for providing network-based functionality within. At step 950, based on the inspection in step 940, the system determines whether the PC document 110 has the appropriate network-based functionality therein. If the system determines that the launched PC document 110 does not have the requisite network-based functionality, then the PC document is opened with no intervention from the PCA extender 120, as shown in step 960.

On the other hand, if the system determines that the launched PC document 110 has the requisite network-based functionality therein, then the network-enabling code 630 is located by the PCA extender 120 in step 965. If the network-enabling code 630 is encrypted, then the PCA extender 120 causes decryption of the network-enabling code 630 to read it, as shown in step 970. At step 980, the system initializes and launches the PC document 110 along with the network-based functionality in accordance with the activation of the network-enabling objects 130 and/or the instructions provided in the network-enabling code 630.

FIG. 10 provides an exemplary illustration of a client/server architecture utilizing the network-enabling features provided to the client, in accordance with the present invention. A client 1000 hosts the underlying PCA 100. The PC document 110 is launched within the PCA 100. As noted above, the PC document 110 is provided with one or more network-enabling objects 130, which communicate with the PCA extender 120 running with the PCA 100.

The client connects to a network 1050 for receiving the network-based functionality for which the network-enabling objects 130 are configured. Using the network 1050, the client 1000 may connect with one or more network servers, such as a web server extender 1060 which runs on a generic web server 1065, or a specialized server(s) 1070 and/or one or more generic third-party servers 1080 for real-time feeds. In one implementation, the client 1000 may be provided with one or more communication means for connecting and interacting with the network 1050. The communication means is coupled to a central processor on the client machine. The scope of the present invention is not limited by the choice of communication means employed. Accordingly, it is possible to use one or more modems, a DSL, an ISDN, a cable modem or any similar device to connect to the server and communicate therewith.

The clients are able to connect to the server and receive their network-based functionality. For example, a client 1000 may connect to a web server extender 1060 which connects to a generic web server 1060 for receiving regular feeds about fluctuations in the stock market, or connect to the specialized server(s) 1070 that provide specialized services to the client, such as real-time feeds 1085, collaboration support 1090, data access 1095, workflow servers, licensing servers and the like. If a client connects to the generic third-party servers 1080 for real-time feeds, the client system is provided with specialized network-based functionality in the PC document 110 that allows the extended PC application 100 to utilize the real-time feeds within the PC documents 110.

It should be noted that the client/server architecture of FIG. 10 can be utilized to download individually or collectively the PCA 100, the PC document 110, network-enabling software 210, and/or the network-enabling object 130 from the Internet or some other compatible

network. For example, a request for a download may be received at the web server extender 1060 which appropriately transmits it to the web server 1065, or the request may be received at the specialized server(s) 1070. Accordingly, the client may transmit a login request, and then make a request to download the desired component(s) of the present invention. The request is appropriately handled at the receiving server, and the request is processed by the server. Finally, the server serves the requested component(s) of the present invention to the remote client. According to one embodiment, the client may be required to authenticate its identity prior to the server serving the requested component.

The web server extender 1060 may be utilized for custom request handling of client requests. According to one embodiment, the custom request handling may be authenticating the remote client. According to another embodiment, the custom request handling may be deciding the version of the requested component(s) to be served to the remote client.

For example, web server extender 1060 may provide intelligence to determine appropriate version of a financial data sheet to return to a client – either a PCA spreadsheet version or an HTML web page version.

FIGS. 11A-11B provide an illustration of the present invention where PC documents are provided with network-based functionality therein. It should be noted that the PC documents described in FIGS. 11A-11B are merely two embodiments of the present invention, and various other embodiments may be constructed using the disclosed invention.

As shown in FIG. 11A, the PC document 1100 relates to a financial spreadsheet with complete information of a user's finances and comprises various components as part of its content. At the top of PC document 1100, there is a page title 1105, which is a part of the static data. The content further comprises summary chart 1110, which contains information regarding

a plurality of financial accounts held in the user's name. Each financial account is listed individually, and may be provided as a link to another document, web site, page within the same document, and/or application to provide users with more detailed information for each entry.

The content may also include formulas that compute mathematical values, such as the one shown as the total current value 1120. According to the present invention, various charts providing graphical depictions 1125 of the data shown in the summary chart 1110 may be provided as part of the content of the PC document 1100.

According to the example depicted in FIG. 11A, each of the financial accounts are provided as a link to another account accessible over the network. Thus, clicking on E*TRADE takes users to the appropriate page that maintains information about the user's stock portfolio; clicking on Citibank Checking takes users to the appropriate page that maintains information about the user's bank account; clicking on Citibank Visa takes users to the appropriate page that maintains information about the user's credit card, and similarly other links transport the user to the appropriate page.

FIG. 11B provides an illustration of a PC document 1130 that contains a plurality of network-based functionalities therein for providing information regarding the user's stock portfolio. As shown in FIG. 11B, the PC document 1130 relates to a financial spreadsheet with complete information of a user's stock portfolio. The PC document 1130 comprises a wireframe, population data, and form data. The wireframe includes static content and native script code that is generally not modified during runtime within a PCA residing on an end-user's system.

Population data is dynamically generated data used to populate specified areas within a wireframe during the PCA's runtime. Generally, population data is supplied by a server and is designated "read only" and is not altered. Population data may be used to populate fields,

strings, and pick lists for display to the user, or transparently as calculation data operated on by the application. Population data may also be defined to be a function of information maintained in the associated wireframe, user input, or both, and may typically be generated or retrieved upon actions such as startup, connection, form submission, or in real-time in response to remote data changes.

Form data is data that is specified or manipulated at the client computer by users during runtime for the eventual purpose of transmission to a server. Form data is usually one-way data transmitted from client to server. But in some circumstances it may be desirable to read such data from a server (for example, as cached prior information to pre-populate a user form or as smart defaults for a user form) to pre-fill forms within a wireframe. Form data includes data items, such as data entered into text box prompts (for example, asking for a user's username and password for authentication) and parameterized data entered into defined queries (such as stock ticker symbol as a parameter in a we b query).

The PC document 1130 comprises a market chart 1135 that is connected to a network so that it receives historical data for the market index in real time. The market chart 1135 forms a part of the population data that is read-only and cannot be changed by the user. A toggle button 1140 is provided to switch the market chart 1135 between display of historical data for each of the selected indices.

The PC document 1130 further includes a portfolio table 1132 for the user, which contains detailed information regarding the various stocks owned by the user. The portfolio table 1132 has layout definition as part of the wireframe, and includes static data such as the heading "Portfolio", heading for various columns, such as "Symbol," "Current Price ($)", and the like, as well as native script code that is not visible to the user. The native script code

includes instructions for fetching the information regarding the stock's trading volume, stock's price, number of shares owned and the like, based on the identity of the user. Information regarding shares owned by the user represents form data and is transmitted from the client computer to the server. On the other hand, displayed information regarding stock's price is population data and is supplied by the server to the client user; this information is read-only.

Network-enabling objects in the form of Alert Wizard button 1142 and Trade button 1144 are provided in the wireframe. The user may click the Alert Wizard button 1142 to launch a network-based functionality that allows the user to receive appropriate alerts and notifications on a regular basis. The user may click the Trade button 1144 to launch a network-based functionality that allows the user to make stock trades over the network. Behind-the-scenes functionality, e.g. trades, is provided either by the PCA extender 120, network-enabling object 130, or by a custom COM object.

Furthermore, the PC document 1130 includes a web browser 1150 that is embedded in the PC document 1130 in accordance with the present invention. According to the invention, the web browser 1150 is a network-enabling object that brings network-based functionality within the PC document 1130.

FIGS. 12A-12E are screen-shots of a personal computer document 110 augmented with developer tools 320 in accordance with the present invention.

FIG. 12A shows an open PC document 110, which is a spreadsheet in the present example, with developer tools 320 available within the PCA being used. As shown in the FIG. 12A, the developer may decide the appropriate data model for the PC document 110 being created, and/or perform desired operations therein, such as XML queries, web data queries, page scraping, database queries and the like.

FIG. 12B shows the open PC document 110 with developer tools 320 available within the PCA being used. As shown in the FIG. 12B, the developer may decide the type of forms to be created for the PC document 110 being created, and/or choose the information to be provided in the form, such as stock information and the like.

FIG. 12C shows the open PC document 110 with developer tools 320. As shown in the FIG. 12C, the developer may decide the appropriate action for objects therein, such as packing and/or unpacking, deploying web pages and the like.

FIG. 12D shows the open PC document 110 with developer tools 320. As shown in the FIG. 12D, the developer may create widgets for use in the PC document 110, such as a facility to allow chat in the PC document 110.

FIG. 12E shows a workbook behavior tool 1200 as a developer tool for determining and/or controlling the behavior of data and/or other content in a spreadsheet workbook. The behavior tool 1200 allows setting of options to control the environment of the PCA 100 at run-time (e.g., locked areas, scratch sheet areas of the spreadsheet, etc.).

FIG. 13-15 illustrate one exemplary embodiment for a system and method for both developing and accessing a web page as an embedded application that can be produced using an augmented spreadsheet program and an enhanced spreadsheet document. An example of such an enhanced spreadsheet in provided in FIG. 11B.

FIG. 13 is an illustration of a client/server architecture in accordance with one exemplary embodiment of the present invention, which provides a system and method for both developing and accessing a web page as an embedded application that is produced using the spreadsheet program 1300 that has been extended with a network-functionality software 1320 and a network-enabled spreadsheet document 1310. The embedded application web page allows

an end-user to enter a stock's symbol, receive relevant historical data relating to the stock via a web server, and chart the received data within the embedded application web page using the built-in charting facilities of the host spreadsheet application 1300.

The developer's system comprises a developer client computer 1302 that is connected to network 1304 via a communication link 1306. Network 1304 may, for example, be a private intranet, a virtual private network, or the global Internet, and the communication protocol used on the network may be TCP/IP or any protocol layered on top of IP, such as SMTP, FTP, HTTP, etc. In this example, the developer client computer 1302 is provided with a spreadsheet software program 1300, such as Microsoft EXCEL or Lotus 123. The client computer 1302 is also preferably provided with a developer's network-functionality software 1320 that adds tools and/or provides functionality to the spreadsheet software 1300 to facilitate the creation of spreadsheet web pages, and an end-user's network-functionality software 1350 for providing runtime support for the network-based functionality within the spreadsheet documents.

The developer is connected to a web server 1314 via the network 1304. The web server 1314 is connected to the network 1304 via a communication link 1316. Web server 1314 provides access via said network 1304 to one or more web pages 1334 or customized embedded application web pages 1336. The original spreadsheet document is now a custom embedded application 1336. Web pages 1334 or customized embedded application web pages 1336 can be stored locally or any place on any network that the web server 1314 can access.

The end-user's system comprises a client computer 1340 that is connected to the network 1304 via a communication link 1306. The client computer 1340 is provided with end-user's network-functionality software 1350, which may include user tools and a PCA extender.

It should be noted that the developer is provided with tools that may be distinct from those provided to a end-user, because the developer uses the system to create an embedded application web page 1336, while the end-user uses the system to access and interact with that embedded application web page 1336.

In summary, the present system and method relates to a developer at client computer 1302, who identifies an HTML page 1334 to be rendered as an embedded application web page. The developer imports a snapshot of the HTML page 1334 into spreadsheet software 1300 and onto the first sheet of a new spreadsheet 1310. The developer may then modify the web page's layout to improve its appearance so that it is more appropriate to a spreadsheet-based application.

The HTML page being rendered will typically include some content that changes infrequently, if at all, and other data or content that changes in response to user actions, remote data updates, and so forth. For example, a web page maintained by an information provider that displays stock prices information will generally include some information that remains essentially constant, such as the page layout, information provider's name and logo, and other information that requires frequent updating, such as stock prices and performance data. The present invention distinguishes between these "static" and "dynamic" content elements, and manage them in different ways.

The static portions of the HTML Web page are preferably rendered in the spreadsheet version of the page as static data within one or more cells or controls within the UI of the embedded application web page. In one embodiment, static HTML text and images may be incorporated in the spreadsheet document using the base HTML import operation of spreadsheet software 1308, if included.

According to one embodiment, the dynamic portions of the HTML Web page are rendered as one or more named cells, charts, or other controls. The developer links these UI elements to state-information elements that may, for example, be stored in cells on a second spreadsheet page or elsewhere in computer memory. These state-information elements can themselves be linked to one or more queries stored, for example, in cells on a third spreadsheet page or elsewhere in memory or the information model 790. At runtime, these queries are executed and the query results are stored as state information in the linked state-information elements. This state-information is then used to populate the dynamic portions of the spreadsheet web page.

It may also be desirable to include form elements to allow an end-user to enter "form data," such as the stock symbol whose price the end-user wishes to retrieve. These form elements are preferably rendered as one or more cells or other controls, such as entry fields, pick lists, and the like, on the first spreadsheet page 1310. The developer links each such UI form element to one or more state-information elements preferably stored in cells on the second page of the spreadsheet 1320 or elsewhere in memory. Information entered by a user in a form element on the first spreadsheet page is passed back to these linked state-information elements for further processing or transmission to web server 1314.

The developer then refines the embedded application UI by hiding distracting elements that will not be needed by end-users in typical operation of the page 1310, such as secondary scratch sheets and extraneous toolbars. Using tools provided by the invention, the developer may decide to restrict or otherwise control the extent to which an end-user may modify or interact with the new web page. The embedded application web page 1310 is then placed into run mode which may prevent any disruption of the UI by end-users. The completed

enhanced spreadsheet web page 1310 is then posted to web server 1314 as custom embedded web page 1336. This deployment may include providing links to the embedded application web page 1336 from other pages, such as the HTML page 1334, on the same or other web sites.

An end-user at end-user computer 1340 may download the embedded application web page 1336 from web server 1314 by opening the page in spreadsheet software 1360 directly or by its hyperlink, which also causes the page to automatically be opened in spreadsheet software 1360 as the enhanced spreadsheet document 1370. The network-functionality software 1350 and the runtime code in the enhanced spreadsheet document 1370 cooperate to execute any queries to retrieve population data elements as specified by the developer. The contents of these elements are used to populate the dynamic data regions of the first spreadsheet page 1370, which may then be displayed to the end-user.

While viewing the enhanced spreadsheet document 1370, the end-user has available all functionality provided by spreadsheet software 1360 that the web page developer has left accessible. For example, if the displayed page contains raw stock data, the end-user may use the spreadsheet software's chart functionality to create a chart from said data. In addition, a scratchpad area may be provided on an additional sheet of the spreadsheet that allows the user to perform arbitrary calculations and manipulations based on the data displayed on the web page. The end-user may also link to data on the web page from a second spreadsheet or other program running on end-user computer 1340. Furthermore, if the user enters a new stock symbol to be charted, the relevant data will be returned to the client via a population data query and charted, causing only the updated areas of the client-side embedded web page 1370 to be updated. As a result, the present system provides high update and refresh performance. It should be noted that custom analysis can be performed locally as opposed to constant calls to the network.

FIG. 14 provides a provides a detailed flow of the operation by a developer to create a web page in accordance with the present invention, using a spreadsheet application 1300 that has been provided with a network-functionality software 1320. In step 1402, the developer launches spreadsheet application 1300. Launching the spreadsheet application 1300 also launches the network-functionality software 1320, including the developer tools and user tools. In this embodiment, developer tools of the network-functionality software 1320 add a developer's toolbar to spreadsheet software 1300 for inserting macros and UI elements into the spreadsheet document 1310. These tools may be made available on one or more developer's toolbars or menus that become embedded in the software's UI.

In one embodiment, the toolbar may include a "new Web page" button for creating a new spreadsheet web page by providing a rendering of a web page as an enhanced spreadsheet document. In step 1404, the developer clicks on the "new Web page" button. In step 1406, the system presents to the developer a wizard containing a palette that displays a selection of templates for designing a web page. In addition, the wizard includes a "browse to web page" button.

According to the present invention, it is assumed that the developer wishes to create a spreadsheet web page that "mirrors" an existing browser-based page, rather than create a spreadsheet web page from scratch. Accordingly, in step 1408, the developer clicks on the "browse to web page" button on the developer toolbar. In step 1410, a browsing window pops up within the spreadsheet's UI and the developer browses to and selects the web page that the developer wishes to copy. According to one embodiment, the developer browses to a web page 1334 displaying stock price performance. An example of such a web page can be seen at http://quote.yahoo.com.

In step 1412, a snapshot of the selected web page is imported onto the first sheet of the spreadsheet document 1310. This step may employ the native HTML import functionality of the spreadsheet application 1300.

As those skilled in the art will recognize, the native HTML import functionality of spreadsheet programs typically create layout errors, including overlapping and obscured items. Consequently, in step 1414, the developer may be provided with a tool for cleaning up any layout problems created by the format translation. Alternatively, the developer tools in the network-functionality software 1320 may be provided with its own HTML import functionality that does not introduce these layout problems. However, it should be noted that the developer laying out a spreadsheet web page should preferably visualize the page in terms of the optimal layout for the spreadsheet application environment, rather than attempting to copy exactly the look and feel of the page's browser-based version. However, this should preferably also be balanced with the desire to maintain as consistent a look and feel between the two versions as possible.

In step 1416, the developer defines state information associated with the embedded application web page 1310. Each state-information element is implemented as a named property, using developer tools provided to create and manage the properties. These elements may be stored as data values in spreadsheet cells on a second spreadsheet page that is accessible to the developer but may not be accessible to the end-user. Alternatively, state information may be stored elsewhere in memory or the information model 790 and referenced only by name. Using the state-information manager, the developer defines individual properties of the state information element and optionally specifies a data type, such as string, numeric, dollar, date, and the like, and a default value for the property. Once defined, the document code

in the spreadsheet document 1310 and/or network-functionality software 1350 allows the element to be referenced elsewhere in the application by property name.

In step 1418, the developer uses the developer tool that is provided to define one or more queries for retrieving dynamic data at runtime. These query specifications may be static, i.e., contain no variables, or may alternatively be a function of one or more named state-information elements. These queries and query results may also be located in cells on a third spreadsheet page or alternatively stored elsewhere in memory.

A query manager may be provided as a developer tool, for facilitating the creation and management of these queries. When the developer wishes to define a query, the query manager creates a new data item and displays a dialog box that prompts the developer to enter the desired query. For example, if the developer wishes to create a query that retrieves the stock price information, the developer might enter the URL query that looks like http://host.cnbc.com/jetson/Chart.html?Symbol=%stocksymbol%+data=raw and specify that data returned by the query should be stored in a data item, that may be labeled *StockData*.

In the above query, the "%stocksymbol%" sub-string indicates that the current value of the *StockSymbol* state-information element should be substituted into the query before it is executed at runtime. This symbol substitution and name reference capability is provided to the end-user by user tools in the network-functionality software 1350. The "data=raw" sub-string indicates to the web server that it should return the raw data, possibly encoded in an HTML table. At runtime, the query will be executed by end-user network-functionality software 1350 and the returned data stored in the *StockData* data item, which may then be referenced by name in other fields and controls in spreadsheet document 1310.

In addition, the query manager may be adapted to automatically create a query to retrieve data identified by the user. According to one embodiment, the query manager may permit a developer to direct a browser to a web page and select, for example, a field on the web page with a mouse. The query manager may then automatically formulate an appropriate query for retrieving data through end-user tools in the network-functionality software 1350.

In step 1420, the developer links dynamic-data and form-data values and other fields on the first spreadsheet page to particular state-information or user interface elements. For example, the developer may link the *Stock Symbol* property defined above to the entry field on the spreadsheet web page. It should be noted that the system may be adapted to allow these elements to be referenced by name in formulas and macros so that the developer need not be aware of where or how they are stored.

In step 1422, the developer creates the spreadsheet application-based chart that will replace the static image used in the original HTML page, which is generated during the import step 1412. In particular, the developer deletes that static image from the first page of the spreadsheet document 1310, selects the area of the page that the chart should occupy, and launches the chart wizard of the spreadsheet application 1308. In the wizard, the developer selects one of the standard line or area charts and then specifies the data source for populating the chart, which in this case is the *StockData* data item described above. This connects the chart to the dynamic data stored in the *StockData* data item. The developer then sets the chart's display properties, such as scaling, labels, axis details and so forth, as desired.

In step 1424, the developer may lock or otherwise disable portions of the UI that will be presented to end-users when spreadsheet document 1310 is opened during runtime. During this step, the developer selects areas of the first spreadsheet page that should be read-only

and presses a "freeze" button on the developer's toolbar. This locks the selected portions of the page so that they cannot be selected and manipulated by the end-user. Also during this step, the developer may specify spreadsheet pages as invisible to the end-user. In the present embodiment, the developer will typically designate only the first spreadsheet page as visible. In addition, the developer may wish to hide any standard spreadsheet toolbars that are inappropriate for the particular spreadsheet web page.

In step 1426, the developer presses a mode toggle button provided on the developer toolbar; causing the end-user tools to take over the spreadsheet application and enforce the embedded application web page behavior that has been designated by the developer. The developer may then work locally on the client machine with the web page in runtime mode to verify that the embedded application web page functions properly.

In step 1428, the developer posts the spreadsheet application web page to the web server 1314 and provides a link to the page to allow end-users to access it. For a more seamless and integrated experience, the developer may additionally utilize a redirector module that runs on the web server in response to end-user requests for the stock web page and transmits either the browser version or spreadsheet version of the web page to a particular end-user based on any of a number of possible criteria.

FIG. 15 provides a detailed flow of the operation by a user to use a web page created in accordance with the present invention. In step 1502, the end-user clicks on or otherwise opens the hyperlink associated with the spreadsheet web page 1336. This hyperlink may be a direct link to the enhanced web page 1336 or a link that invokes redirector software at Web server 1314. In the latter case, the redirector software can return either the browser web page 1334 or the enhanced spreadsheet page 1336 based on either client capabilities – namely,

whether or not spreadsheet application 1300 and end-user network-functionality software 1350 are available on client machine 1340 – or on preferences for the specific end-user. Such preferences may be based on the type of enhanced document or on the particular page or on the web site as a whole or upon other controlling criteria. The page is transmitted to end-user client machine 1340 in step 1504 and automatically opened within spreadsheet software 1360. Where enabled, this operation may open the spreadsheet application within the launching browser container using available systems mechanisms such as Active Document Servers and multiple-document interfaces (MDI).

In step 1506, end-user network-functionality software 1350 and runtime code in the downloaded embedded application web page 1370 cooperate to execute any queries in the web page and to store the returned data in one or more population data elements. In step 1508, this state information is used to populate any designated dynamic-data regions of the first spreadsheet page. In step 1510, the complete "web page" is displayed to the end-user in the PCA format. As noted above, while viewing the page, the end-user has access to all built-in spreadsheet functionality of spreadsheet software 1360 allowed by the developer. As a result, the user can still do cell analysis, refer to cells, create charts and figures, and the like.

In some implementations of this embodiment, end-user network-functionality software 1350 and runtime code in web page 1336 may cooperate with a web server extension 1330 (FIG. 13) to periodically re-execute a query or otherwise update the dynamic data used to populate the web page. For example, a "time until stale" field may be associated with each query included in the spreadsheet web page. When this time expires, end-user network-functionality software 1350 determines whether an appropriate network connection still exists for updating the dynamic data associated with the query. If the connection still exists, the query

is run again, and the dynamic data displayed on the web page is updated with the latest query results. Alternatively, web server extension 1330 may detect that data associated with a registered query in enhanced document web page 1370 has changed and proactively push updated query results to the document for viewing at the client.

In some implementations of this embodiment, the developer constructing embedded application web pages may find that two or more pages at a given Web site (or, for that matter, all pages at a given site) are related and appropriate to be rendered as embedded application web pages. In this case, a spreadsheet version of each of the two or more pages may be created and stored within a single enhanced spreadsheet document, on distinct spreadsheet sheets. These separate sheets may be linked such that navigating from one to another merely results in the display of a different sheet in the same enhanced spreadsheet document. This capability of storing multiple web pages within a single document provides the ability to improve interoperability and integration between what were originally disconnected pages. It also provides the ability to cache, snapshot, or transmit entire web sites or portions of web sites, and work within an entire web site while disconnected from the hosting network, after having downloaded the single enhanced spreadsheet document containing the multiple embedded application Web pages, and also caching snapshots of any needed population data query results.

As the discussion relating to FIGS. 13-15 illustrates, one embodiment of the present invention enables the development of web pages, web applications and even web sites that can be accessed directly within non-browser client applications, such as word processors, spreadsheet programs, graphics applications and the like. The invention exploits both the power of the containing application, for content manipulation, calculation capabilities and the like, and distributed and web-based functionality typically available only through a separate web browser.

FIG. 16 is an illustration of a client/server architecture in accordance with another exemplary embodiment of the present invention, which may be used to provide a system and method that facilitate the cooperative writing and editing of newspaper articles and other written works.

In this embodiment, there are three distinct users interacting with the system, with uses and functions. The first user may be a reporter or staff writer who is responsible for authoring news stories for publication. The writer uses a client computer 1602 having a word processing software program 1608 such as Microsoft WORD or Corel WORDPERFECT. Writer computer 1602 may also provided with a network-functionality software 1610 that enhances the functionality and communications capability of word-processing software 1608.

The second user may be an editor on the news staff, who uses a client computer 1632 that is provided with the word processing software 1608 and the same network-functionality software 1610 that the writer has on his or her client machine 1602.

The third user may be a software developer who might be employed by, or a consultant to, the newspaper. The developer uses client computer 1642 which may be provided with developer software comprising word processing software 1608, a developer add-in 1650, as well as end-user network-functionality software 1610. This software facilitates the creation of special-purpose word-processing documents that exhibit enhanced functionality and communications capabilities.

The client/server comprises a server computer 1628. The server computer 1628 is preferably provided with appropriate software to support and manage various system services provided to the client computers 1602,1632,1642 through a web server 1314 with one or more web server extensions 1330.

The various client computers, and the web server 1314 are typically connected to a network 1304, via respective communication links 1306 and 1316. This network may be any of the various network types described previously, including for example LAN, VPN, intranet, extranet, or the public Internet.

FIGS. 17-19 provide a detailed flow of the operation by the writers, editors and developers of the system in accordance with the present invention.

Generally, a developer at developer computer 1642 uses word processing program 1608, including the developer add-in 1650 and the network-functionality software 1610, to create an enhanced document-template file 1612. The enhanced document-template file includes word processing formatting defaults that satisfy the requirements for article submissions specified by the newspaper's editorial board and is used as a template for creation of new articles. In addition, the document-template file preferably includes macros and visible elements and objects for providing enhanced functionality to an end-user, such as a writer working at writer client computer 1602. This enhanced document-template file may be stored at server computer 1628 for web-based access by staff writers.

A writer wishing to prepare an article for publication downloads the document-template file to writer computer 1602 and opens the file within word processing software 1608. The writer is then presented with an enhanced UI, which includes one or more non-standard toolbars for accessing particular services from within the UI of word processing software 1608. For example, the toolbar may include an icon for conducting database or archive searches via network 1304. Some or all of these services may be provided with the aid of server computer 1628.

The enhanced document-template user interface also preferably includes a workflow control button for changing the state of the document and notifying members of the editorial staff of the pending draft. Clicking on a workflow button may also cause the system to perform other tasks concurrently, such as time-stamping and archiving a copy of the submission.

An editor, working at the editor computer 1632, may then open the submission in editing software 1608 and review it. The UI of editing software 1608 is also enhanced to provide additional functionality and workflow control capabilities through the user tools in the network-functionality software 1610. Using these workflow control capabilities, the editor may return the article to the writer for further revisions or forward it to another entity within the system, such as the typesetting department.

A more detailed description of system operation as it relates to a developer is now provided in connection with the flow chart of FIG. 17. In step 1702, the developer launches word processing software 1608, including developer add-in 1650 and the user tools which provide tools for inserting formatting information and runtime code into a document-template file. These tools may be made available on a developer's toolbar that is part of the software's UI. As a result, word processing software 1608, with the developer add-in 1650 and the user tools provides a powerful development platform for quickly and easily generating document-template files 1612 using simple tasks such as "drag and drop" and text insertion, as described in more detail below.

In step 1704, the developer creates a new document-template file 1612 and inserts appropriate text formatting settings and content. The format defines the overall layout of the file as it will be displayed to the end-user (e.g., a writer) and specifies the number and type of fields that must be filled in before a writer may submit an article (e.g., byline, editor name, deadline

and the like). As noted above, formatting requirements for article submissions will typically be specified in advance by the newspaper's editorial board, and, as a result, the designer will often have little discretion in defining these features. Thus, for example, a "choose template" button on the developer's toolbar may pull up a menu of predefined formats that have been approved by the editorial board. The developer may then choose one, which is automatically inserted in the document-template file.

In step 1706, the developer adds additional functionality to the enhanced document-template file 1612. For example, the developer may wish to add a workflow-control button to the file so that, when the button is clicked by an end-user (such as a writer) at runtime, it will automatically post the file to a specified group of end-users (such as one or more editors).

In this embodiment, adding functionality of this type to a document-template file may be facilitated by a tool available on the developer's toolbar. For example, the developer's toolbar may comprise a "workflow" button that can be dragged from the toolbar and dropped into the document at any desired location. A dialog box may then pop up that prompts the developer to specify details of the communication such as the identity of the editor who is to receive the file when the writer clicks on the workflow button. Alternatively, some or all of this information may be left blank to be supplied by the writer at runtime. For example, the developer may add to the document-template file a pop-up menu with a complete list of editors. At runtime, the writer may simply choose the appropriate editor to receive draft submissions from the menu list.

In step 1708, the developer may insert dynamic data queries into the document-template file. As described below, these data queries are executed at runtime by an end-user computer (such as writer computer 1602) to return population data that is inserted into

the document-template file for display to the end-user, and such queries can be made to execute either when the document is opened or in response to some action by an end-user such as clicking a button in the enhanced document. The developer's tools may preferably include a tool for facilitating the creation of such queries and associating them with the appropriate event in the document-template file.

In step 1710, once the developer is satisfied with the document-template file, he or she saves it on server computer 1628 from where it may be downloaded by a writer to client computer 1602.

A more detailed description of system operation as it relates to a writer is now provided in connection with FIG. 18. In step 1802, the writer launches word-processing software 1608 (including user tools) on the writer client computer 1602.

In step 1804, the writer determines whether he or she has previously downloaded an appropriate document-template file for the article to be written. If the appropriate document-template file has been previously downloaded, the writer opens the file within word-processing software 1608. Otherwise, the writer downloads the appropriate document-template file from, for example, server computer 1628, and opens it within word-processing software 1608.

In step 1806, network-functionality software 1610 and/or runtime code in the opened document-template file 1612 cooperate to present an enhanced UI and various functionalities to the writer. In this exemplary embodiment, the document is enhanced in at least the following ways. First, it displays the document in a format that conforms to the requirements set by the newspaper's editorial board. Second, it executes queries in the document-template file, and populates fields within the document with the results of those queries. Third, it provides

the writer with one or more non-standard toolbar buttons or menus for accessing network-based services from within word-processing software 1608. These toolbars may be embedded in the usual toolbar section of UI of the word processor program 1608 (e.g., along the top of the UI), or may alternatively float at other locations within the UI.

The toolbar may facilitate the end-user access to a wide variety of services. For example, the toolbar may include a "search" button for triggering an archival or Internet search. In particular, when the writer clicks on this button, a dialog box may pop up that prompts the writer to enter a search term or string. Alternatively, the document may exhibit context-sensitive behavior, such as the user may be able to specify search terms by highlighting text within the open document. The dialog box may also request that the writer specify the locations to be searched. Thus, for example, the writer may limit the search to the newspaper's archives or specify that the search should include the World Wide Web. This information may be forwarded by writer computer 1602 to a search program on server computer 1628 or other computer. Search results may be returned to writer computer 1602 for display to the writer in a window within the enhanced UI of word-processor software 1608. Displayed results may be dragged directly from the window into the word processor document.

In another embodiment, the UI may include a "template checker" button that will determine whether any specified document meets the predefined formatting requirements specified by the newspaper's editorial board.

The toolbar may also include a "review pending articles" button that allows a writer to view a list of, for example: (1) articles currently under development, (2) favorite articles, (3) articles posted to the writer by other writers for review, (4) or any other personalized list of articles.

The toolbar may also include a "live collaboration" button that launches a window to provide live collaboration on a document with a specified group of end-users. The collaboration session may be managed by appropriate software on the server computer 1628.

In addition, the UI may include a "submit draft" button for transmitting a completed draft to an appropriate editor or editors, as described in more detail below.

In step 1808, the writer begins the drafting process. During this process, the writer may, for example, type text into the document using a standard keyboard, or may copy text (e.g., quotes from a press release) from another document and drop the text into the document. In addition, the writer may employ the "search" functionality described above to identify additional archived or other material that he or she wishes to incorporate into the article.

Once the writer finishes the drafting process, in step 1810, the writer clicks on the "submit draft" button. As noted, this causes a copy of the document to be forwarded to an appropriate editor or editors. In addition, in some embodiments, clicking on the "submit draft" button may cause other concurrent actions to be performed. For example, the system may transmit an e-mail notification to the editor or editors that a document is being sent to them for their review. In addition, the system may automatically timestamp and archive a copy of the draft submission on server computer 1628.

A more detailed description of system operation as it relates to an editor is now provided in connection with FIG. 19. As shown in Fig. 19, in step 1902, the editor receives notification that a draft has been received that requires the editor's review. In one embodiment, this notification may be received by e-mail.

In step 1904, the editor launches editing software 1608 (including software 1610). In step 1906, the editor opens the received submission. In this exemplary embodiment, network-

functionality software 1610 and the code within the received submission may cooperate to provide the editor with an enhanced UI that is analogous to the enhanced user interface described above. It should be noted, however, that an editor will typically be provided with a plurality of workflow buttons due to their privileges in the workflow system, rather than the single "submit draft" button described above. More specifically, for the editor the UI dynamically provides both a "return to writer" button for returning the document to the writer with revisions and comments, and a second button for forwarding the document to another location (e.g., the typesetting department) when the article is ready for publication.

Furthermore, in some embodiments, it may be desirable to provide the editor with additional workflow buttons. For example, it may be desirable to provide an editor with a "legal department" button that transmits a copy of the document to the legal department for their review before publication. When an editor clicks on the "legal department" button a message box may pop up to allow the editor to explain the reason for forwarding the article (e.g., libel concerns).

In step 1906, the editor edits the article. During editing, the editor may revise the document and add comments or questions to it. This editing and change management handling is provided by the word processing software, illustrating the advantages over attempting such functionality out of a browser-based application.

Once editing is completed, in step 1908, the editor determines whether the article is ready for publication or requires additional work by the author. If the article is ready for publication, then, in step 1910, the editor clicks on an appropriate workflow button that forwards the article to another entity within the system, such as the typesetting department, as noted above. Otherwise, in step 2212, the editor clicks on a second workflow button that returns the

article to the writer for further drafting. This workflow process between the editor and writer may be repeated until the article is deemed ready for publication.

As is made apparent in this example, this invention enables the development of an entirely new class of distributed applications that expose and integrate both corporate and external services and resources while running within familiar desktop applications. The resulting enhanced functionality can be packaged and exposed to end users as enhancements to the containing application itself, as elements of embedded applications embodied in enhanced documents, or a combination of the two.

FIG. 20 is an illustration of a client/server architecture in accordance with another embodiment of the present invention, which may be used to provide a system and method that allows an illustrator or another author to create, distribute, and license illustrations, documents, and other works of authorship.

As shown in FIG. 20, the system comprises a client computer 2002 belonging to an illustrator. This illustrator computer 2002 is provided with an illustrator software program 2008 such as Adobe ILLUSTRATOR. The Illustrator computer 2002 is also preferably provided with a licensing add-in 2010 that enhances the functionality and communications capability of illustrator software program 2008. The Illustrator computer 2002 is connected to a network 1304, such as the Internet or a private intranet, via a communications link 1306.

The system shown in FIG. 20 further comprises a licensing web server computer 2014 that is connected to network 1304 via a communications link 1316. Licensing server 2014 is provided with a licensing management software program 2030 that manages licensing of illustrations created with illustrator computer 2002, as described in more detail below. Licensing server 2014 is typically owned and operated by an entity in the business of managing licensed

works (a "licensing manager"). It should be noted that the system enables easy integration with existing server-based services and functionality. In this particular scenario the system could alternatively enable integration of an existing licensing system 2028 with the Web and specifically with enhanced documents through the use of licensing server 2014 and adapter web server extension code 2030.

The system also includes a consumer interacting with a client computer 2015. For the purposes of this embodiment, the consumer wishes to browse and possibly purchase illustrations created by the illustrator. Consumer computer 2015 is connected to network 1304 via a communications link 1306. Consumer computer 2015 is provided with a viewer software program 2018 that facilitates viewing of illustrations 2012 that may be downloaded via network 1304 or accessed by other means such as CD-based distribution. In addition, consumer computer 2015 is further provided with an end-user add-in 2020, which allows the consumer to view and purchase licensed illustrations created with illustrator computer 2002, as described in more detail below.

Typical operation of the present system and method will be described in detail below with reference to the flowcharts of FIG. 21-22. Generally, an illustrator at illustrator computer 2002 purchases one or more "document license controls" from the licensing manager that operates licensing web server 2014. Each document license control entitles the illustrator to certain licensing management services from licensing server 2014 for a designated illustration 2012. When the illustrator designates an illustration for licensing, the system registers the illustration with licensing computer 2014 and embeds licensing control code in the illustration document 2012 that, with the user licensing tool 2020, controls the document's subsequent use and distribution.

A consumer who downloads the illustration 2012 from, for example, the illustrator's Web site, may open the illustration within viewing software 2018, possibly being presented (by viewer add-in 2020) with a dialog box that sets forth licensing terms, restrictions, and other information. Alternatively, if viewing is unrestricted the consumer may not be apprised of any such licensing restrictions until attempting to execute a licensed function such as copying or printing the illustration. If the consumer wishes to print a copy of the illustration, the transaction will be confirmed with the consumer and a connection is established with licensing computer 2014, which electronically charges the consumer the appropriate licensing fee. Printing would then commence at client computer 2015 having been allowed by end-user licensing add-in 2020.

A more detailed description of system operation as it relates to an illustrator is now provided in connection with the flowchart of FIG. 21. As shown in FIG. 21, in step 2102, the illustrator launches illustrator software 2008 (including the licensing tool 2010). In step 2104, the illustrator creates an illustration 2012 at illustrator computer 2002 using illustrator software 2008.

In step 2106, the illustrator determines whether he or she possesses a desired number of "document license controls" from the licensing manager that owns licensing web server 2014. As noted above, each document license control entitles the illustrator to certain licensing management services from licensing server 2014 for a designated illustration.

If the illustrator concludes that he or she has an adequate number of document license controls, flow proceeds to step 2110. Otherwise, in step 2108, the illustrator purchases a desired number of document license controls from the licensing manager. This transaction may,

for example, be conducted directly through licensing server 2014 maintained by the licensing manager.

In one embodiment, the first time that the illustrator purchases a document license control, he or she receives a copy of licensing tool 2010, typically downloaded to illustrator computer 2002 from licensing computer 2014 via network 2004. The licensing tool 2010 adds a toolbar to illustrator software 2008 that includes an icon for licensing particular illustrations, as well as icons for displaying other licensing information, as described in more detail below. It should be noted that such special-purpose add-ins and toolbars are generally enabled by this system, and that a developer of such add-ins could use programming APIs and runtime support from generic add-ins and other code at either the client or server to provide easy access to specialized functions and services.

Once the illustration is complete, in step 2110, the illustrator determines whether he or she wishes to allocate one of the purchased document license controls to this illustration 2012, i.e., whether he or she wishes the licensing manager to manage the distribution and licensing of this illustration. If the illustrator does not allocate a document licensing control to this illustration, the flow ends at step 2112. Otherwise, flow proceeds to step 2114.

As noted above, licensing tool 2010 displays an additional toolbar within the UI of illustrator software 2008. One of the icons on this toolbar is preferably a "license illustration" icon, which the illustrator may click on when he or she wishes to allocate a document license control to a particular illustration. Other buttons might be used, for example, to display account information for the illustrator with the license provider or to display specific license information for a viewed document that was licensed previously by the illustrator.

Thus, in step 2114, the illustrator (having decided to use a document license control for this illustration) clicks on the "license illustration" icon. This operation triggers Licensor Authentication step 2116 whereby the licensing tool 2010 displays a dialog to verify the illustrator's identity. In step 2118, a secure connection, perhaps using secure HTTP, is established between illustrator computer 2002 and licensing server 2014 via network 2004 and the identity of the illustrator is conveyed to the licensing server. In step 2120, licensing computer 2014 transmits various pieces of information to illustrator computer 2002. For example, in this exemplary embodiment, licensing server 2014 may transmit the number of document license controls that the illustrator has remaining. In addition, licensing server 2014 may transmit updated status information concerning licensing of the illustrator's other works (number of licenses, dollar value of licensing fees, etc.). As mentioned previously, additional icons on the toolbar may be provided to permit the illustrator to view this latter information.

In step 2122, licensing tool 2010 causes a dialog box to pop up that displays information about the document to be licensed, such as its name, type, and size. The dialog box may also display additional information such as the number of remaining document license controls available to the illustrator (received in step 2120 above).

Also on this licensing dialog box, a pick list or other selection mechanism is provided which displays to the illustrator the available license types, fee structures, and other license terms. The illustrator may then specify those license terms in step 2122, as desired. In an exemplary embodiment, one menu choice may be "standard license," which may implicitly specify a plurality of set terms that have been predefined by the licensing manager, the illustrator, or a combination of the two. Once the license terms have been specified, the illustrator may click on a "license now" button to indicate his or her approval (step 2124).

Clicking on the "license now" button causes a transaction to be executed with licensing server 2014 that consumes one of the illustrator's document license controls. In particular, in step 2126, a notification is transmitted from illustrator computer 2002 to licensing server 2014 that includes information concerning the document to be licensed. In step 2128, licensing server 2014 creates and registers a document key for the illustration that includes an identifier for the illustration and associated licensing terms. In step 2130, licensing server 2014 subtracts one from the number of document license controls available to the illustrator.

In step 2132, licensing tool 2010 receives the new license information and embeds licensing control code within the illustration document 2012, converting it into an enhanced illustration document. In an enhanced embodiment, this licensing control code may comprise a plurality of locked and write-protected macros that enforce the licensing restrictions specified by the illustrator. Additionally, the licensing tool 2010 may cause the illustration to be encrypted or otherwise protected such that the illustration cannot be accessed without licensed use with an end-user license tool 2020 in viewer software 2018. The document is then ready for publication. Thus, in step 2134, the illustrator publishes the illustration in one or more locations, such as on his or her web site.

A more detailed description of system operation as it relates to a consumer is now provided in connection with the flowchart of FIG. 22. In step 2202, the consumer launches viewer software 2018 on consumer computer 2015. In step 2204, the consumer directs a browser to the illustrator's web site. In step 2206, the consumer finds an illustration of interest and downloads a copy of it (including the embedded licensing control information) to consumer computer 2015 via network 2004. In step 2208, the consumer views the downloaded illustration with viewer software 2018. Alternatively, the end-user could navigate to the publishing web site

using a browser and open the illustration directly, causing viewer software 2018 to be opened automatically at the client (with add-in 2020), possibly within the same containing window as the browser itself.

In step 2210, add-in 2020 and the licensing control code within the enhanced illustration document 2012 cooperate to display a dialog box indicating that the illustration is licensed through the license manager. In this embodiment, this dialog box may further comprise a license term button that the consumer may click to see the license terms associated with this illustration. In this embodiment, license terms for the illustration may be structured so that the consumer may download and view the illustration at no cost.

In step 2212, the consumer examines illustration 2012 and determines whether he or she wishes to print a copy. If not, the flow ends at step 2214. Otherwise, the flow proceeds to step 2216, where end-user licensing tool 2020 causes a print dialog to be displayed that permits the consumer to set printing details. In some embodiments, this dialog may also remind the user of any licensing fees. Once the consumer is satisfied with the printing details, he or she may click a "Print" button (step 2218).

In step 2220, add-in 2020 authenticates the consumer and then establishes a secure HTTP session with licensing Web server 2014, as specified within the licensing code embedded in the enhanced illustration itself. In step 2222, an electronic transaction is conducted between consumer computer 2015 and licensing server 2014 in which the consumer is charged the licensing fee for the illustration. This charge may involve collecting charging information via a secure web interface or could execute automatically using known charging information for a frequent consumer. Then, in step 2224, the document is printed for the consumer.

According to the present invention, the illustrator or licensor is able to restrict content that the consumer can access. For example, when the licensor is licensing content, he or she it can license the content incrementally. For example, the consumer may be allowed to view a synopsis section of the PC document 110, but not the entire PC document 110 until payment is made by the consumer. As another example, a consumer may be allowed to view a thumbnail of an image, but not the whole image until a certain date has passed. The mechanism for allowing such a licensing involves providing the content 615 as partially packed and partially unpacked. As a result, when it is desired that the user not be able to access certain network-based functionality, the content 615 will not get completely unpacked at runtime.

In addition, it is also possible to provide application control using the licensing technology described above. As a result, the PCA 100 can be limited or controlled according to licensing rules employed. For example, a consumer may be able to see a full image but the printing capabilities of the PCA 100 may be disabled, so the consumer cannot print the image. As another example, the consumer may be allowed to view a PC document 110 but not be allowed to collaborate on the PC document 110 until payment is received. An exemplary mechanism to allow such licensing is to not allow the unpacking of a part of the network-enabling code 630, which deprives the consumer of the underlying functionality.

As is made apparent by this example, the invention provides for integration of new functionality into existing desktop applications such that end-users, as opposed to developers, can easily create and interact with enhanced documents that leverage a variety of distributed services (such as the exemplary distributed licensing service), and that it provides the means for existing service providers to broaden their availability, utility, and overall value through greatly expanded access and availability.

The described architecture for the method and system of the present invention provide a wide array of both client- and server-side benefits. It enhances the functionality available to an end-user by facilitating connection to network-based services (e.g., Web browsing, conferencing, chat, e-mail, licensing, DRM, and workflow), and making these services available within familiar and powerful PCAs. Additionally, because the enhanced PCA is "network aware," it can provide the end-user with distinct on-line and off-line usage models. For example, the augmented desktop application may, during on-line usage, provide access to network services and, during off-line usage, allow the end-user to work on previously downloaded content. In addition, data entered by the end-user during off-line usage may be stored for later upload and subsequent processing by a server (i.e., synchronization).

Furthermore, the PCA with the network-based functionality enhances the UI provided to both developers and users. This enhanced UI may include non-standard toolbars and buttons for launching special-purpose tools and executing commands as described above.

In addition, because the enhanced application can connect to one or more remote servers, the system may efficiently allocate tasks between client and server. For example, a server may be tasked with collecting, processing, and formatting data required by an enhanced application. Enhanced pipes may be established between server and client to facilitate downloading of this information. Both clients and servers may also intelligently pre-fetch content and data in order to respond to anticipated requests by an end-user. The ability to replicate a Web site within a PC document, and install such a PCA-based site on a client machine, can improve performance because only real-time dynamic data (i.e., population data) must be obtained over the Internet from the server. Because the present system provides for the separation if desired of wireframe and population data, the wireframe may be treated as a

versioned and infrequently downloaded independent element. This decreases bandwidth requirements with use and improves the end-user experience by decreasing screen-refresh times. It also enables more efficient caching of multiple instances at the client because the wireframe is cached only once. Custom, licensed, or public-domain differencing techniques may be employed to exploit this aspect of the present invention. In addition, this separation may be extended to multiple population data blocks per page, each of which can be managed and updated independently.

Furthermore, the present system facilitates optimal workload balancing between client and server. In many cases, it may be desirable to assign various background and agent-type tasks to the server. For example, the server may be programmed to monitor a Web site for changes, and notify the client when a change is detected. Document control (e.g., managing who has permission to modify a document), workflow processes, and PCA-based conferencing may also be centrally managed by a server. The server may be programmed to detect obsolete versions of runtime code or wireframes on the client and update that code, either automatically or under the end-user's explicit control.

The above architecture facilitates both unstructured use as well as more structured use. Structured use denotes the concept of a packaged application. If one restricts a user's operations, the PCA has transformed into a new more specialized application, which leverages the desired functionality of the original PCA. The invention can also be used in unstructured formats, typically in the form of user tools. The user tools will provide quick utilities to enhance the power of the PCA, such as a screen scraper. Thus, the invention is useful in the format of a highly structured application, quick unstructured utilities, and everything in between.

End-users are also preferably provided with access to, and some control over, runtime utilities via the above architecture. For example, end-users may initiate conferences and chat sessions with each other from within their augmented desktop-application environments.

Furthermore, since numerous modifications and variations will readily occur to those skilled in the art, it is not desired that the present invention be limited to the exact construction and operation illustrated. Accordingly, all suitable modifications and equivalents which may be resorted to are intended to fall within the scope of the claims.